Scheduling with time dependent discrepancy times

Florian Jaehn · Helmut A. Sedding *

This is a post-peer-review, pre-copyedit version of an article published in Journal of Scheduling (2016) 19:737–757. The final authenticated version is available online at: http://doi.org/10.1007/s10951-016-0472-2

Abstract In time dependent scheduling, various processing time functions are studied. Yet, absolute value functions have surprisingly been omitted from the discussion. Such a processing time function increases linearly on a job's discrepancy from its ideal midtime. The objective is to find a schedule that minimizes the makespan. This introduces the Discrepancy Time Minimization Problem. This single machine scheduling problem with time dependent processing times is motivated by the optimization of walking times at a car assembly line. Its decision version is NP hard, as we show by reduction of the Even Odd Partition Problem. For the variant of a known start time, we develop several heuristics. Further insights form lower bounds and dominance rules for a branch and bound search. Numerical experiments show the performance of our algorithms on problem instances of up to 60 jobs. For the variant with a common ideal midtime and a flexible start time, we present a polynomial time algorithm.

Keywords Time Dependent Scheduling; Nonmonotonic Piecewise Linear Processing Time; Convex Processing Time; Single Machine Scheduling; Assembly Line Worker Path Minimization

1 Introduction

We consider a problem we encountered in production planning for car assembly lines. The given production facility consists of a walkable conveyor, moving steadily along a straight line, and shelves, placed statically along the conveyor. We focus on a single worker, working at one car. The worker is responsible for a set of assembly jobs at the car. To gather required material, the worker needs to interrupt each job and walk to its respective shelf aside the conveyor, then walk back. The next job also requires material, from the same or another shelf, and so forth for the remaining jobs. We assume that only the base length (productive assembly time) of each job is constant. In contrast, the walking time of each job is time dependent. If we manage to place each job right at the time when the conveyor passes the

Florian Jaehn · Helmut A. Sedding *
University of Augsburg

* corresponding author

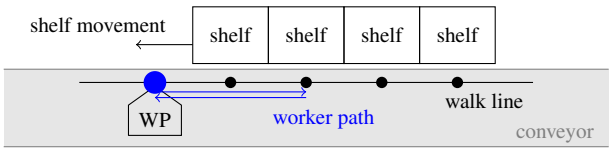

Fig. 1 Shelves are located directly, within an insignificant gripping distance, aside the conveyor. Therefore, the worker only walks along a straight line in one dimension, along the conveyor direction. Furthermore, he or she remains atop the conveyor at all times. For ease of walk time calculation, the coordinate system is placed on the conveyor. Then, the shelves move relative to the conveyor along a straight line. The worker is standing at the working point (WP). To pick up material for a job $j \in J$, the worker walks atop the conveyor to the pick point at the shelf, which moves with the shelf. This point is reached at the time m_j. Picking takes no time in our model. Thus, after picking, the worker proceeds to walk from the exact same point back to the working point.

job's shelf, the walking times become minimum. Given a set of jobs, assigned to a set of shelves, our objective is to find a schedule that minimizes the makespan.

The worker starts at a known start time at the working point on the conveyor. There, he or she executes all jobs in set J sequentially, in an order that is to be determined. For each job $j \in J$, the processing time p_j is a variable that depends on the time the job is executed. It consists of a constant work time, specified by base length l_j, and a variable walk time. In our application, the work time is split into two parts. The first part is the preparatory work. Then, the worker walks to pick up the material, and, as the second work part, assembles it. The duration of these parts may be different in each job. We assume an even split, dividing the base length l_j into two equal halves.

The walk time to a shelf depends on the conveyor position. The conveyor speed is constant. As the worker exclusively walks atop the conveyor, we base the worker's reference coordinate system on the conveyor. By this change of viewpoint, the conveyor becomes static, and everything else moves in relation to the conveyor. Hence, the shelves appear to move along the conveyor (which in turn is static). The worker walks with a constant, uniform velocity in all directions upon the conveyor. Hence, distances are of the same length in all directions. Therefore, if the worker walks from his working point to a certain other point on the conveyor, the way back has exactly the same length, in both time and space. This holds regardless whether this point is forwards or backwards on the conveyor line. We apply this principle for calculating the walk time to a shelf and back. Because the shelf moves relative to the conveyor, the additional difficulty is to find out the point in time m_j, for a job $j \in J$, when the worker actually reaches the shelf. By the previous argument, both walks to a point and back take the same time. Hence, the start and completion time are equidistant from m_j. Also, m_j represents the middle between start and completion time of the walk. Therefore, we call m_j the *midtime* of a job $j \in J$. Note that, by prepending and appending the work time halves $l_j/2$, we do not alter the requirement that m_j represents the midtime of a job. In the following, we calculate the actual walk time.

Shelves are placed right at the conveyor. We take the simplifying assumption that the distance, when measured orthogonally to the conveyor line, is insignificantly small, within gripping distance. Therefore, no walk is necessary when the worker's working point on the conveyor is right in front of the shelf. Following that, the worker only needs to walk along a straight line, which is coincident with the conveyor direction. Hence, the walk is measured in only one dimension. This is visualized in Figure 1. The time when the shelf is right at the working point of a job $j \in J$ is measured when the midtime m_j equals M_j, for an ideal

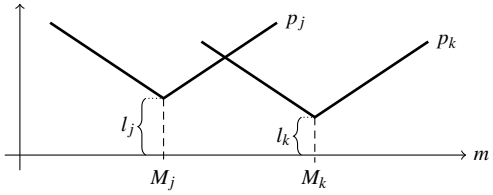

Fig. 2 The processing time $p_j = l_j + a\,|m_j - M_j|$ of a job j consists of the base length l_j and, scaled by $a \in (0,2)$, the difference between the ideal midtime M_j and the midtime $m = m_j$ it is actually scheduled at. Depicted are two job's processing times p_j, p_k, each with a different base length (l_j and l_k), and a different ideal midtime (M_j and M_k).

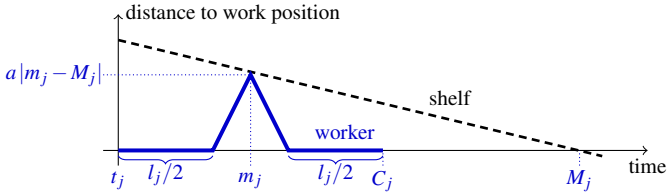

Fig. 3 In the graphic, the abscissa displays the time. The ordinate displays the distance of an object (here: a shelf) to the work position on the conveyor. It is measured in time units scaled by $a \in (0,2)$. The factor a is constant and depends on conveyor and worker speed. Relative to the conveyor, the dashed line represents the position of a shelf over time. The solid line represents the worker position in time. The time span between start time t_j and completion time C_j represents the processing time p_j for a job j. It consists of two halves of the base length l_j, and, in between, the walk time to the shelf. The shelf is reached at time point m_j. It is apparent that the time for walking to the shelf equals the time for the way back. Therefore, the worker always meets the shelf at the middle of the job's processing time. Hence, as a result, $m_j = (t_j + C_j)/2$.

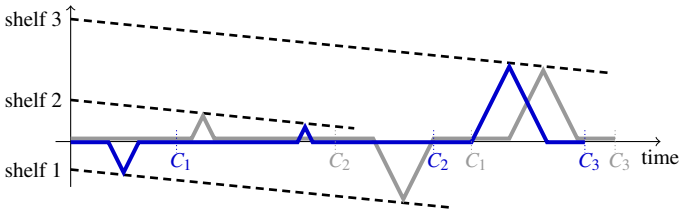

Fig. 4 This figure shows a worker's position in two different job sequences, dark blue and light gray. Multiple jobs are scheduled sequentially without overlaps and without idle times. The processing time of a job depends on the time it is executed. Hence, the sequence (or ordering) of jobs determines $C_{\max}$. As in Figure 3, each dashed line represents the movement of a shelf. From bottom to top, the lines respectively belong to job 1, 2, and 3. In dark blue depicted is the movement of a worker who processes the jobs in sequence $1 \to 2 \to 3$. In light gray, it is $2 \to 1 \to 3$, which takes longer because the total walk time is longer.

midtime M_j which encodes the shelf position. Thereby, M_j represents the point in time in which the shelf passes the working point. A negative value for M_j means that already at the (global) start time, the shelf is behind the working point. The walk distance from and back to the work point is an absolute value function of the difference between work point and the moving shelf position. Hence, we factor in the conveyor and the worker speeds (v_{worker} and v_{conveyor}) to obtain a slope $a = 2 \cdot v_{\text{worker}}/v_{\text{conveyor}}$ (factor 2 for two ways), called *growth*

factor, of the walk time. We can safely assume that the worker walks faster than the conveyor moves, thus $0 < a < 2$. Then, the walk time is calculated by the term $a|m_j - M_j|$. We further add the two base length halves. This results in the processing time $p_j = l_j + a|m_j - M_j|$. This notion is depicted in Figures 2 and 3. Note that p_j is defined in terms of the midtime here, but it is possible to formulate this in terms of the job's start time, as we show in section 3.

At the known start time, the worker starts to process one job after the other. Neither overlaps, nor idle times are allowed. Hence, a job starts right after completion of the predecessor, if there is one. Its processing time is decided according to its start time. Then, the sequence (or ordering) of jobs iteratively determines all processing times. This is visualized in Figure 4. It is required to minimize the completion time of the last job.

The paper is organized as follows: First, we review and discuss related work (section 2). Then, we define the Discrepancy Time Minimization Problem setting in several variants (section 3). Then, we describe essential properties for the variant with a single common ideal midtime (section 4). These results are used to prove the NP hardness of the decision versions of the problem and two subproblems (section 5). For the variant with a common ideal midtime with no given start time, we present a polynomial time algorithm (section 6). In section 7, we present several algorithms to solve the variant with a given start time. We present a mixed integer programming (MIP) and a dynamic programming (DP) formulation. Basing on the complexity proof, we develop several lower bounds. The DP algorithm is the base for several dominance rules. Heuristics deliver an initial upper bound. These results culminate in a branch and bound (B&B) search and its truncated, heuristic version, the TrB&B. Computational experiments show each algorithm's performance in a new testbed for this problem (section 8).

2 Related Work

The problem we study is related to classical scheduling with objectives that regard earliness/tardiness or barely tardiness, to time dependent scheduling with piecewise linear processing time functions, and to applied research in the optimization of the car assembly.

The nonmonotonic piecewise linear processing time function reminds one of the objective function in the early/tardy scheduling problem. In three-field notation, it is described by $1 \,||\, \sum |C_j - d_j|$, where each job j has a given due date d_j and, assigned by the schedule, a completion time C_j. With this objective, each job should ideally be placed close to its due date. When deviating from the due date, the costs increase linearly. This is similar to our problem. Here, each job should also be placed close to a certain point in time (the ideal midtime). On deviation, not the costs though, but the processing time increases linearly as well. Hence, in both problems, this deviation linearly increases the respective objective. The major difference to our problem is that in the early/tardy scheduling problem, the processing times are constant. Therefore, the order of scheduling a subset of jobs does not influence their sum of processing times. In our problem however, the processing time of each job is determined by its start time. Hence, the sum of processing times of a subset of jobs is variable, as it additionally depends on their sequence. This property is revealed when, e.g., setting the start time of a certain job to the completion time of the preceding jobs. To optimize this completion time, it is not sufficient to know the optimum set of preceding jobs. Instead, for this set, it is required to know the optimum sequence. The early/tardy scheduling problem is NP hard, as Garey et al. (1988) show by reduction of the Partition Problem. Even more, Wan and Yuan (2013) recently show NP hardness in the strong sense, by reduction of the Three Partition Problem. Even the common due date variant is still NP hard (Hall et al.,

1991). The known fastest exact solution methods for the early/tardy scheduling problem are described by, e.g., Bülbül et al. (2007); Sourd and Kedad-Sidhoum (2008); Sourd (2009); Tanaka et al. (2009). They all use time-indexed formulations, which are relaxed to construct lower bounds. However, to apply this method to our problem, one would need to divide the discretization step by a to avoid rounding errors. However, this looks impractical. For only halving the discretization step, or equivalently doubling the processing times, Bülbül et al. (2007) report that computation times increase fivefold. This indicates that these algorithms are not directly applicable to our problem.

Another problem with a piecewise linear objective function is the tardiness problem $1 \mid\mid \sum \max\{0, t_j - d_j\}$. Note here, the function is monotonic. This problem is still ordinarily NP hard (Du and Leung, 1990), but not strongly, as there exists a pseudopolynomial algorithm (Lawler, 1977). For common due dates $d = d_j$, the shortest processing time (SPT) rule yields an optimum solution (Lawler and Moore, 1969). If each job's tardiness is weighted, Kellerer and Strusevich (2006) and Kacem (2010) describe fully polynomial time approximation schemes (FPTAS).

Piecewise linear start time dependent scheduling problems are studied before. To our best knowledge, no one considers nonmonotonic processing time functions yet. But, there is a trail of work on monotonic functions. Scheduling with job deterioration first gathered attention when Browne and Yechiali (1990) introduced a time dependent scheduling problem. Their stochastic model, if viewed as deterministic, condenses to $1 \mid p_j = l_j + a_j t_j \mid C_{\max}$. Here, each job j is given a base length l_j and a growth factor a_j, while a schedule sets the job's start time t_j. They show that $C_{\max}$ is minimized when sorting the jobs nondecreasingly by l_j/a_j. However, in other applications, jobs deteriorate only after a certain critical date d_j, remaining in a good condition beforehand. This model, which can be denoted by $1 \mid p_j = l_j + a_j \max\{0, t_j - d_j\} \mid C_{\max}$, was introduced by Kunnathur and Gupta (1990). They provide a branch and bound algorithm and several heuristics. Wu et al. (2009) develop several dominance rules and a lower bound for a branch and bound algorithm; they also state two heuristics. Kononov (1997) proves NP hardness by reduction of the tardiness problem, also for the case with a common $a = a_j$. For the case with a common critical date $d = d_j$, Kubiak and van de Velde (1998) show NP hardness by reducing the partition problem and present a heuristic, a branch and bound algorithm, and a pseudopolynomial algorithm. Cai et al. (1998) discover a fully polynomial time approximation scheme. Kubiak and van de Velde (1998) generalize this problem by introducing a bounded deterioration case, which stops any processing time increase when starting a job after a common maximum deterioration date $D > d$ which may be infinite. They present two pseudopolynomial algorithms. Kovalyov and Kubiak (1998) describe a fully polynomial time approximation scheme. Cheng et al. (2003) look at an inverse problem: decreasing processing times until a common due date d, precisely $1 \mid p_j = l_j - a_j \min\{d, t_j\} \mid C_{\max}$. They prove NP hardness by reduction of the Partition Problem, and present a pseudopolynomial algorithm as well as several heuristics. For the case of $2da_j \leq l_j$ for all j, Ji and Cheng (2007) develop a fully polynomial time approximation scheme. Several reviews of time dependent scheduling problems exist (Alidaee and Womer, 1999; Biskup, 2008; Cheng et al., 2004; Janiak et al., 2011), and we may especially refer to Gawiejnowicz's (2008) in-depth treatise.

Optimization problems that are related to scheduling problems are found, e.g., in vehicle routing. A variant with only one vehicle is called traveling salesman problem. Here, the distance between two cities maps to, e.g., a sequence dependent job processing time on a single machine. In time dependent scheduling, the processing time depends not on the sequence but on the point in time that the job is executed. Variable distances which depend on the node arrival time are introduced by Malandraki and Daskin (1992). They name it time

dependent traveling salesman problem, although a definition with the same name existed before (Picard and Queyranne, 1978). Also noteworthy is a vehicle routing problem with time windows and a time-varying congestion where the distances depend on a piecewise linear function of time by Ahn and Shin (1991). They recognize that the distance function needs to fulfill the non-passing-property, which requires that it is not possible to overtake a vehicle, thus the later a vehicle starts, the later it arrives. In our work, we do also have a piecewise linear function for which the non-passing-property holds, as the worker is faster than the conveyor. Ahn and Shin (1991) extend insertion, savings, and arc exchange heuristics known before. Exact solution procedures are not discussed, as the main difficulty was to find feasible instances at all. In our problem, the main difficulty is not to find feasible solutions (all permutations of jobs are feasible), but to know that a solution is optimum.

On the application side, there exists a long thread of work about assembly line balancing (Boysen et al., 2007). In the classic sense, this problem is to optimize the assignments of tasks (here: jobs) to work stations. Each station has a certain cycle time which must not be exceeded. As a second constraint, each job has space requirements, e.g., for material placement. Therefore, there are assembly line balancing problems which consider both time and space constraints within a station (Bautista and Pereira, 2007; Chica et al., 2012). Following this, to minimize worker walk times at each station, Boysen et al. (2015) identify line side placement as a relevant planning problem, which concerns about positioning bins within a station. This problem is addressed from an operations research view by Klampfl et al. (2006). In our practical application, we consider the bin or shelf positions as given, and, accordingly, change the sequence of jobs. Bukchin and Meller (2005) consider bin placement on a more aggregate view of the whole assembly line, also respecting fill rates. To incorporate walk times in the assembly line balancing procedure, one may turn to solution strategies which take into account sequence-dependent setup times (Andrés et al., 2008; Martino and Pastor, 2010; Scholl et al., 2013). Although these setup times could be used for material pickings, the movement of the conveyor belt would then be neglected. To still ensure feasibility of the result, a safety factor would need to be added to all walk times. The more accurate it is possible to estimate the walk times, the smaller the safety factor can be. Therefore, in this work, we establish a model for time dependent walk times which is able to factor in the moving conveyor. The walk time can amount to a significant part of the cycle time (Boysen et al., 2015). Therefore, it is important to better estimate and, consequently, reduce walk times.

3 Problem Definition

The formal definition of our practical problem considers distinct ideal times, representing distinct shelf positions. A subproblem is stated with a common ideal midtime; in this case there is only one shelf. In section 5, we show that this case is NP hard. Also, we define a slightly generalized problem, where the start time is a variable. There, the objective is to minimize the makespan, which is the time span between start and completion of the schedule. This problem is NP hard as well, as we will show. Interestingly, this generalized problem in combination with only one common ideal time turns out to be polynomially solvable (section 6).

Definition 1 (Discrepancy Time Minimization Problem) Given a growth factor $a \in (0,2)$ and a set of n jobs J, where each job $j \in J$ is given a base length $l_j \in \mathbb{R}_{\geq 0}$ and an ideal midtime $M_j \in \mathbb{R}$.

A schedule (S,t) comprises the job sequence, which is a bijective function $S : J \mapsto P$ that assigns each job a position in set $P := \{1,\ldots,|J|\}$, and a start time $t \in \mathbb{R}$. From a

given schedule, the values for all midtimes and all processing times are derived. For this, the start time t_j and the completion time C_j of each job $j \in J$ is calculated. The start time of the first job equals the start time t, hence $t_{S^{-1}(1)} = t$; the start time of each remaining job equals the completion time of the preceding job, hence $t_{S^{-1}(i)} = C_{S^{-1}(i-1)}$ for $i = 2,\dots,n$. The completion time C_j of a job $j \in J$ is calculated by

$$C_j = C_j(t_j) := \begin{cases} t_j + \dfrac{l_j - a(t_j - M_j)}{1 + a/2} & \text{if } t_j \le M_j - l_j/2, \\ t_j + \dfrac{l_j + a(t_j - M_j)}{1 - a/2} & \text{if } t_j > M_j - l_j/2. \end{cases} \tag{1}$$

Then, a job's midtime is $m_j = (t_j + C_j)/2$. A job's processing time is stated by $p_j := l_j + a\,|m_j - M_j|$. The last completion time is $C_{\max}(S,t) := C_{S^{-1}(n)}$.

The objective is to find a schedule (S,t) that minimizes the makespan, denoted by $Q(S,t) := C_{\max}(S,t) - t$. This problem is stated as $1\big|p_j = l_j + a\,|m_j - M_j|\big|$*makespan*, we abbreviate this by **fDTMP** (where 'f' indicates the flexible start time).

The **DTMP** problem sets a fixed start time t, with $t = 0$ if not further specified. It is stated as $1\big|p_j = l_j + a\,|m_j - M_j|\big|C_{\max}$. Here, the makespan equals $C_{\max}$.

We denote the subproblems where all jobs are given the same common ideal midtime $M = M_j$, $j \in J$, as **cDTMP** and **cfDTMP**, respectively (where, 'c' indicates the common ideal midtime).

The decision version of each problem variant asks, for some given value $q \in \mathbb{R}_{\ge 0}$, whether there exists a schedule with $Q(S,t) \le q$.

One first insight is: $p_j = l_j \iff m_j = M_j$. Furthermore $p_j > l_j \iff m_j \neq M_j$. In all cases, $p_j \ge l_j \ge 0$. Note that the definitions of t_j, C_j, m_j, and p_j ensure that $p_j = l_j + a\,|m_j - M_j| = C_j - t_j$.

The allowance of idle times would only increase the makespan $Q(S,t)$, because $C_j(t_j)$ is monotonically increasing with t_j, as $\frac{d}{dt_j} C_j(t_j) \ge \frac{2-a}{2+a} > 0$. Thus, the DTMP disallows idle times between jobs. Hence, $Q(S,t) = \sum_{j \in J} p_j$.

Note that we allow for negative start times, especially in the fDTMP. This is necessary, as the start time shall not impose any restriction on the scheduling of the jobs. Hence, negative values for t_j, C_j, and m_j may occur.

4 Problem Properties

In this section, we present properties of the cDTMP, and later the cfDTMP.

4.1 cDTMP Properties

First, we look at properties of cDTMP instances. Thus, all jobs have the common ideal midtime M, and a schedule starts at a given time t. If M is before or at t, i.e. $M \le t$, we are actually able to express the makespan in a closed-form formula. Figure 5 visualizes this situation.

Definition 2 Given a cDTMP instance with n jobs and a schedule (S,t). For any $x, y \in \mathbb{R}$, define $f(x) := \left(\frac{2+a}{2-a}\right)^x$, $g(x) := \frac{2}{2-a} f(x)$, and $Q_0(S,y) := y\,f(n) + \sum_{j \in J} l_j\, g(n - S(j))$.

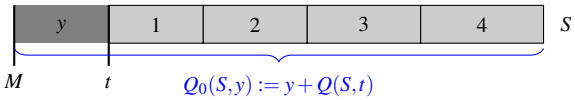

Fig. 5 Given a set of jobs J with a common ideal midtime M starting at $t \geq M$, and a sequence S. Then, $y = t - M$. Property 2 introduces the closed-form expression $y + Q(S,t) = Q_0(S,y) = y\,f(n) + \sum_{j \in J} l_j\, g(n - S(j))$.

Property 1 Given a cDTMP instance and a schedule (S,t) with tM. Let $y = t - M$ and $n = |J|$. Then, $y + Q(S,t) = Q_0(S,y) = y\,f(n) + \sum_{j \in J} l_j\, g(n - S(j)) = y\,f(n) + Q(S,M)$.

Without loss of generality, assume $J = \{1,\ldots,n\}$ and $S(j) = j$ for $j \in J$.

Proof For ease of description, we express the duration y by a virtual job 0, which starts at M and completes at t, with processing time $p_0 = y$. Thus, the job set extends to $J' = \{0\} \cup J = \{0,1,\ldots,n\}$. Therefore, we define schedule (S',M) with $S'(j) = j+1,\ j \in J'$. Now, job 1 starts at $t = M + p_0$.

We want to know the value of $Q_0(S,y) = y + Q(S,t) = Q(S',M) = \sum_{j=0}^{n} p_j$. For this, we need to know the value of p_j for all $j \in J'$. We begin with the definition $p_j = l_j + a\,|m_j - M_j|$. Knowing that $M_j = M$ and $m_j \geq M$, we simplify it to $p_j = l_j + a(m_j - M)$. We then express $m_j - M$ in terms of the job's start time t_j:

$$m_j - M = p_j/2 + t_j - M \tag{2}$$

$$= \begin{cases} p_j/2 & \text{if } j = 0 \\ p_j/2 + C_{j-1} - M & \text{if } j > 0 \end{cases} \tag{3}$$

$$= p_j/2 + \sum_{k=0}^{j-1} p_k. \tag{4}$$

Thus,

$$p_j = l_j + a(m_j - M) \tag{5}$$

$$\iff p_j = l_j + a\left(\frac{p_j}{2} + \sum_{k=0}^{j-1} p_k\right) \tag{6}$$

$$\iff \left(1 - \frac{a}{2}\right) p_j = l_j + a \sum_{k=0}^{j-1} p_k \tag{7}$$

$$\iff p_j = \frac{2}{2-a}\left(l_j + a \sum_{k=0}^{j-1} p_k\right). \tag{8}$$

Comparing p_j to p_{j-1} for $j \in J$, we observe:

$$p_j - p_{j-1} = \frac{2}{2-a}\left(l_j - l_{j-1} + a \sum_{k=0}^{j-1} p_k - a \sum_{k=0}^{j-2} p_k\right) \tag{9}$$

$$\iff p_j - p_{j-1} = \frac{2}{2-a}(l_j - l_{j-1} + a p_{j-1}) \tag{10}$$

$$\iff p_j - \frac{2+a}{2-a} p_{j-1} = \frac{2}{2-a}(l_j - l_{j-1}). \tag{11}$$

This expresses p_j as a recurrence relation, with the base case $p_0 = y$. We convert this to a closed-form expression. Let, using (8), $l_0 = \frac{2-a}{2} y$. Let $\Phi_j = \frac{p_j}{f(j)}$. From (11), for $j \in J$,

$$\frac{p_j}{f(j)} - \frac{\frac{2+a}{2-a} p_{j-1}}{f(j)} = \frac{\frac{2}{2-a}(l_j - l_{j-1})}{f(j)} \tag{12}$$

$$\iff \frac{p_j}{f(j)} - \frac{p_{j-1}}{f(j-1)} = \frac{\frac{2}{2-a}(l_j - l_{j-1})}{f(j)} \tag{13}$$

$$\iff \Phi_j - \Phi_{j-1} = \frac{\frac{2}{2-a}(l_j - l_{j-1})}{f(j)}. \tag{14}$$

This recurrence begins with $\Phi_0 = p_0 = y$, hence

$$\Phi_j - \Phi_0 = \sum_{k=1}^{j} \Phi_k - \Phi_{k-1} \tag{15}$$

$$\iff \Phi_j - P_0 = \sum_{k=1}^{j} \frac{\frac{2}{2-a}(l_k - l_{k-1})}{f(k)} \tag{16}$$

$$\iff \frac{p_j}{f(j)} = y + \sum_{k=1}^{j} \frac{\frac{2}{2-a}(l_k - l_{k-1})}{f(k)} \tag{17}$$

$$\iff p_j = y f(j) + \frac{2}{2-a} \sum_{k=1}^{j} (l_k - l_{k-1}) f(j-k). \tag{18}$$

This is the closed form expression for p_j. We transfer this result to calculate $Q_0(S,y)$. We append a virtual job $n+1$ with $l_{n+1} = 0$. When stating the processing time of job $n+1$, we observe that with (8), $p_{n+1} = \frac{2a}{2-a} \sum_{j=0}^{n} p_j = \frac{2a}{2-a} Q_0(S,y)$. Therefore, using (18),

$$\begin{aligned}
Q_0(S,y) &= \frac{2-a}{2a} p_{n+1} \\
&= \frac{2-a}{2a} y f(n+1) + \frac{1}{a} \sum_{j=1}^{n+1} (l_j - l_{j-1}) f(n+1-j) \\
&= \frac{1}{a} \left(l_0 f(n+1) + \sum_{j=1}^{n+1} (l_j - l_{j-1}) f(n-j+1) \right) \\
&= \frac{1}{a} \sum_{j=0}^{n} l_j \left(f(n-j+1) - f(n-j) \right) \\
&= \frac{1}{a} \sum_{j=0}^{n} l_j \left(\frac{2+a}{2-a} - 1 \right) f(n-j) \\
&= \sum_{j=0}^{n} l_j \frac{2}{2-a} f(n-j) \\
&= l_0 \frac{2}{2-a} f(n) + \sum_{j=1}^{n} l_j g(n-j) \\
&= y f(n) + \sum_{j=1}^{n} l_j g(n-j).
\end{aligned}$$

Furthermore, $Q_0(S,y) = y f(n) + Q_0(S,0) = y f(n) + Q(S,M)$. □

As the processing time grows symmetrically both before and after the ideal midtime, Q_0 can be used symmetrically on schedules that complete before a common ideal midtime M.

4.2 cfDTMP Properties

In the following, we consider the cfDTMP. This problem variant is closely related to optimum early/tardy scheduling with a nonrestrictive (i.e., large) common due date, thereby allowing idle time before the first job, as in Kanet (1981). There, any optimum schedule adheres to a *V-shaped* ordering. This means that all jobs that complete before (after) the due date are sorted nondecreasingly (nonincreasingly) by processing time. Garey et al. (1988) present a proof on the optimality of such a V-shape, and introduce a number of further properties of the problem. We show similar properties in the cfDTMP (Property 3-7), although the proofs are more elaborate because of the variable processing time. First we define several terms, which are visualized in Figure 6.

Definition 3 Given a cfDTMP instance with a job set J with the common ideal midtime M. For schedule (S,t), we define set $A = \{j \in J \mid m_j < M\}$ consisting of jobs before M and set $B = \{j \in J \mid m_j > M\}$ consisting of jobs after M.

Note that there might exist a job $j \in J$ where $m_j = M$, which thus is neither in A nor in B. If such a job exists, we denote it as χ.

We also define an index $A : \{1,\dots,|A|\} \mapsto J$ on A, as well as an index on B. The index on A represents the order of the actual midtime such that for $i, j \in \{1,\dots,|A|\}$ and $i < j$, we have $m_{A(i)} > m_{A(j)}$. Similarly for $i, j \in \{1,\dots,|B|\}$ and $i < j$, we have $m_{A(i)} < m_{A(j)}$.

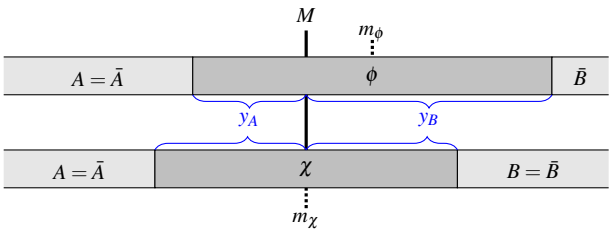

Fig. 6 If existing, for job χ there is $m_\chi = M$. If χ is nonexistent, the notation defines job ϕ which begins before and completes at or after M.

The tasks for a given schedule, suppose χ exists, are then indexed as follows:

$$A(|A|),\dots,A(2),A(1),\chi,B(1),B(2),\dots,B(|B|).$$

Even if there is no χ, the jobs obviously center around M in an optimum schedule. In any case, there must always exist a job, denoted as ϕ, that lies over M: Define job ϕ with $t_\phi < M \leq C_\phi$. Define the overlaps before and after M as $y_A := M - m_\phi + p_\phi/2$ and $y_B := m_\phi - M + p_\phi/2$. Note that the overlaps are both nonnegative and $y_A + y_B = p_\phi$. Job ϕ belongs to either A or B. Further note that if job $A(1)$ completes at M and job $B(1)$ starts at M, then $\phi = A(1)$.

Subsets of A,B that exclude job ϕ are defined as $\bar{A} := A \setminus \{\phi\} \subseteq A$ and $\bar{B} := B \setminus \{\phi\} \subseteq B$.

The makespan $Q(S,t)$ of a given optimum schedule can be split at M into two parts: the makespan before M is $Q_0(\bar{A}, y_A)$, and the makespan after M is $Q_0(\bar{B}, y_B)$. As A and B form a partition of $J \setminus \{\phi\}$, the makespan is $Q(S,t) = Q_0(\bar{A}, y_A) + Q_0(\bar{B}, y_B)$.

We now show that in a solution with minimum makespan $Q(S,t)$, the jobs in A and B follow a specific order.

Property 2 (V-Shape) An optimum cfDTMP schedule (S,t) arranges all jobs in a V-shape, which means for two jobs $A(i), A(j) \in A$ with $i < j$ we have $l_i \leq l_j$, and the same holds for B.

Proof Assume for a contradiction that there is an optimum schedule, which is not V-shaped. Then, this schedule can be improved as follows.

We know $y_A \leq p_\phi$.

The value of y_A is largest if ϕ completes at M. In this case, $y_A = p_\phi$ and $Q_0(\bar{A}, y_A) = Q_0(A, 0) = 0 \cdot f(|\bar{A}|) + \sum_{j=1}^{|A|} l_{A(j)}\, g(|A| - j)$.

Because $g(x)$ is a monotonically increasing function, $Q_0(A,0)$ is minimum if the list of jobs A is ordered nonincreasingly by base length. Thus, for two jobs $i, j \in \{1, \ldots, |A|\}$ and $i < j$ we have $l_{A(i)} \leq l_{A(j)}$.

The case $y_A < p_\phi$ can be visualized by an artificial job $1'$ with $p_{1'} = y_A$ completing at M, thus $l_{1'} < l_1$. As a smaller length for $A(1)$ does not inflict the sort index, the schedule remains minimum.

The same argument follows symmetrically for B. □

Now we show that for ϕ, the actual midtime m_ϕ is M. In consequence, job χ exists. For this, we look at instances with an odd number of jobs only.

Property 3 Given a cfDTMP instance with an odd number of jobs. Then, job χ exists in any optimum schedule.

Proof (by contradiction) Suppose an optimum schedule (S,t) with no job χ, thus $y_A \neq y_B$.

As the total number of jobs $n = 2h+1$ is odd, we have $|A| + |B| = 2h+1$. Thereby, either $|A| < |B|$ or $|A| > |B|$. These two cases will be further split into $y_A < y_B$ and $y_A > y_B$.

Case (1) where $|A| < |B|$ and $y_A < y_B$:

In this case, $\phi = B(1)$, as visualized in Figure 7.

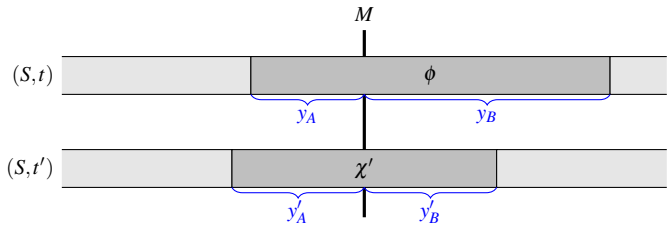

Fig. 7 Proving Property 3, case (1). Schedule (S,t') starts earlier than (S,t), such that ϕ becomes χ'. The makespan of (S,t') is smallest.

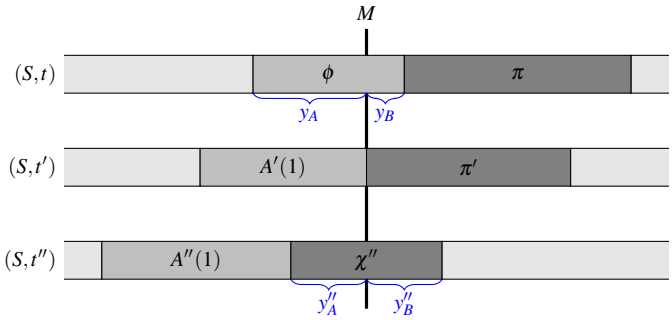

Fig. 8 Proving Property 3, case (2). Schedule (S,t') starts earlier than (S,t), such that π starts at M. Furthermore, schedule (S,t'') starts earlier than (S,t'), such that $\pi' = \pi$ becomes χ''. The makespan of (S,t'') is smallest.

Define schedule (S,t') with $t' < t$ such that $m'_\phi = M$, $A' = A$ and $B' = \bar{B}$. Here, $y'_A = y'_B$. Thus, χ' exists.

As there is no job χ in (S,t), the inequation $l_j < p_j$ holds for all $j \in J$, and we can say that $y'_A + y'_B < y_A + y_B$. Knowing that $|A| < |B| \iff f(|A|) < f(|B|) \iff f(|\bar{A}|) \leq f(|\bar{B}|)$, we show:

$$\begin{aligned}
& y'_A + y'_B < y_A + y_B \\
\iff & y'_A - y_A < y_B - y'_B \\
\iff & f(|\bar{A}|)(y'_A - y_A) < f(|\bar{B}|)(y_B - y'_B) \\
\iff & f(|\bar{A}|)y'_A - f(|\bar{A}|)y_A < f(|\bar{B}|)y_B - f(|\bar{B}|)y'_B \\
\iff & f(|\bar{A}|)y'_A + f(|\bar{B}|)y'_B < f(|\bar{A}|)y_A + f(|\bar{B}|)y_B.
\end{aligned}$$

Now we compare the makespans $Q(S,t)$ and $Q(S,t')$:

$$\begin{aligned}
Q(S,t') &= Q_0(\bar{A}, y'_A) + Q_0(\bar{B}, y'_B) \\
&= f(|\bar{A}|)y'_A + Q(\bar{A}, M) + f(|\bar{B}|)y'_B + Q(B, M) \\
&< f(|\bar{A}|)y_A + Q(\bar{A}, M) + f(|\bar{B}|)y_B + Q(B, M) \\
&= Q_0(\bar{A}, y_A) + Q_0(\bar{B}, y_B) \\
&= Q(S,t).
\end{aligned}$$

Concluding, we see that $Q(S,t') < Q(S,t)$. Therefore, (S,t) is not an optimum schedule.

Case (2) where $|A| < |B|$ and $y_A > y_B$:

In this case, $\phi = A(1)$, see Figure 8.

As $\phi \in A$ and $|A| < |B|$, it follows $|B| \geq 2$. Furthermore, in this case $|\bar{A}| = |A| - 1$, hence $|\bar{A}| < |B| - 1$.

Now we calculate the values of y_A and y_B. As $y_A > y_B$, there is $m_\phi < M$, thus $p_\phi = l_\phi + a(M - m_\phi)$. Therefore,

$$\begin{aligned}
y_A &= M - m_\phi + p_\phi/2 \\
&= M - m_\phi + \big(l_\phi + a(M - m_\phi)\big)/2 \\
&= (a/2 + 1)(M - m_\phi) + l_\phi/2,
\end{aligned}$$

$$\begin{aligned} y_B &= m_\phi - M + p_\phi/2 \\ &= m_\phi - M + \big(l_\phi + a(M-m_\phi)\big)/2 \\ &= (a/2-1)(M-m_\phi) + l_\phi/2. \end{aligned}$$

The difference is $y_A - y_B = 2(M-m_\phi)$, thus $y_B = y_A - 2(M-m_\phi)$.

To show the following inequality, we use $\gamma := 2-a$ and $\delta := 2+a$.

$$\begin{aligned} \delta\,(M-m_\phi) - \delta(M-m_\phi) &= 0 \\ \iff 2y_A - \delta(M-m_\phi) &= l_\phi \\ \iff 4y_A - 2\delta(M-m_\phi) &= 2l_\phi \\ \iff y_A\gamma + y_A\delta - 2\delta(M-m_\phi) &= 2l_\phi \\ \iff y_A\gamma + y_B\delta &= 2l_\phi \\ \iff (2l_\phi/\gamma - y_A)(\gamma/\delta) &= y_B \\ \iff (2l_\phi/\gamma - y_A)\,f(-1)\,f(|B|) &= y_B f(|B|) \\ \iff (2l_\phi/\gamma - y_A)\,f(|B|-1) &= y_B f(|B|) \\ \iff (2l_\phi/\gamma - y_A)\,f(|\bar{A}|) &< y_B f(|B|) \\ \iff (2l_\phi/\gamma)\,f(|\bar{A}|) &< y_A f(|\bar{A}|) + y_B f(|B|). \end{aligned}$$

Define schedule (S,t') with the smallest $t' < t$ such that still $A' = A$ and $B' = B$. Then, $y'_A = p'_\phi$, $y'_B = 0$, and $B' = \bar{B}'$. For the value of p'_ϕ, we mirror $C_\phi(M)$ to the left hand side of M, thus $C_\phi(M) - M = 2l_\phi/\gamma = p'_\phi$.

$$\begin{aligned} Q(S,t') &= Q_0(\bar{A}', y'_A) + Q_0(\bar{B}', y'_B) \\ &= Q_0(\bar{A}, p'_\phi) + Q_0(B,0) \\ &= Q_0(\bar{A}, 2l_\phi/\gamma) + Q(B,M) \\ &= Q(\bar{A},M) + Q(B,M) + (2l_\phi/\gamma)\,f(|\bar{A}|) \\ &< Q(\bar{A},M) + Q(B,M) + y_A f(|\bar{A}|) + y_B f(|B|) \\ &= Q_0(\bar{A}, y_A) + Q_0(B, y_B) \\ &= Q_0(\bar{A}, y_A) + Q_0(\bar{B}, y_B) \\ &= Q(S,t). \end{aligned}$$

Therefore, $Q(S,t') < Q(S,t)$.

Define schedule (S,t'') with $t'' < t'$ such that the actual midtime of job $\pi := B(1)$ is $m''_\pi = M$, thus $\chi'' = j_\pi$ and $y''_A = y''_B = p''_\pi/2 = l_\pi/2$. Also, $B'' = \bar{B}' = B \setminus \{\pi\}$. We compare $Q(S,t'')$ and $Q(S,t')$, using the inequations $l_\pi < p'_\pi$ and $|A| \le |B| - 1$:

$$\begin{aligned} Q(S,t'') &= Q_0(\bar{A}'', y''_A) + Q_0(\bar{B}'', y''_B) \\ &= Q_0(A, l_\pi/2) + Q_0(\bar{B}', l_\pi/2) \\ &< Q_0(A, p'_\pi/2) + Q_0(\bar{B}', p'_\pi/2) \\ &= Q(A,M) + Q(\bar{B}',M) + f(|A|)\,p'_\pi/2 + f(|B|-1)\,p'_\pi/2 \\ &\le Q(A,M) + Q(\bar{B}',M) + f(|B|-1)\,p'_\pi \\ &= Q_0(\bar{A}', y'_A) + Q_0(\bar{B}', y'_B) \\ &= Q(S,t'). \end{aligned}$$

Concluding, we see that $Q(S,t'') < Q(S,t)$. Therefore, (S,t) is not an optimum schedule.

The case $y_A = y_B$ is not needed as it would imply that χ already exists in (S,t). The case where $|A| > |B|$ follows the same scheme as the other cases, with A and B swapped.

Therefore, in all cases, a schedule (S,t) with $m_\phi \neq M$ is not an optimum schedule. □

Property 4 Given a cfDTMP instance with an odd number of jobs. In any optimum schedule, the sets A and B have the same cardinality $|A| = |B|$.

Proof Given an optimum schedule (S,t), we first look at the case $|A| < |B|$. By Property 3, χ exists. Thus $|A| + |B|$ is even. Given that $|A| < |B|$ we deduct that $|A| < |B| - 1$.

We now create a schedule (S,t'') with $t'' < t$, such that $\chi'' = B(1)$ and $A''(1) = \chi$, thereby $|A''| = |B''|$. For this, we now show that $Q(S,t'') < Q(S,t)$.

First, we define schedule (S,t') with $t' < t$, such that $A'(1) = \chi$ completes at M and $B'(1) = B(1)$ starts at M. Thus, χ' does not exist. Still $|A| < |B|$, but the difference lessens to $|A'| = |B'| + 1$.

With schedule (S,t') and (S,t'') defined, we can now follow the argumentation of the previous proof, case (2), but now asserting values $y_A = y_B$. In this way, we can show that $Q(S,t'') < Q(S,t)$.

Therefore, schedule (S,t) cannot be optimum.

The same conclusion can be made for the case $|A| > |B|$ where the argumentation is similar. □

We now look at what happens if jobs $A(i)$ and $B(i)$ are swapped in the sequence S, thereby defining sequence $S_{(i)}$.

Property 5 Given a cfDTMP instance with $n = 2h+1$ jobs $j \in \{0,\dots,2h\}$ and a schedule (S,t) with $|A| = |B| = h$ and $\chi = 0$. For some $i \in \{1,\dots,h\}$, define schedule $(S_{(i)},t_{(i)})$ derived from (S,t) such that $m_\chi = M$ remains, but jobs $A(i)$ and $B(i)$ swap positions: $A_{(i)}(i) = B(i)$ and $B_{(i)}(i) = A(i)$. Then, the makespans of both schedules are equal.

Proof Because $\chi = 0$ and $|A| = |B| = h$ we know that

$$\begin{aligned} Q(S,t) &= Q_0(A, l_\chi/2) + Q_0(B, l_\chi/2) \\ &= l_\chi f(h) + \sum_{j=1}^{h} l_{A(j)}\, g(h-j) + \sum_{j=1}^{h} l_{B(j)}\, g(h-j) \\ &= l_\chi f(h) + \sum_{j=1}^{h} \left(l_{A(j)} + l_{B(j)}\right) g(h-j). \end{aligned}$$

As swapping the summands $l_{A(i)}$ and $l_{B(i)}$ does not affect the result of the addition, we can conclude that $Q(S_{(i)},t) = Q(S,t')$. □

Now the question is, which of the jobs becomes job χ.

Property 6 Given a cfDTMP instance with $n = 2h+1$ jobs $j \in \{0,\dots,2h\}$ where $l_0 = \min\{l_j \mid j = 1,\dots,2h\}$. In an optimum schedule then $\chi = 0$ holds.

Proof As $j = 0$ has the minimum base length, and because we know that in an optimum schedule, such a job is behind all jobs in A, and before all in B, we know that $\chi = 0$. □

We now consider the final property that renders a cfDTMP schedule optimum.

Property 7 Given a cfDTMP instance with $n = 2h+1$ jobs $\{0,\dots,2h\}$, sorted nondecreasingly by base length. Then, for an optimum schedule (S,t), there is $\chi = 0$, $|A| = |B|$, and for $i = 1,\dots,h$ the set $\{A(i),B(i)\} = \{2i,2i-1\}$. The makespan is $Q(S,t) = l_\chi f(h) + \sum_{i=1}^{h}(l_{2i} + l_{2i-1})\, g(h-j)$.

Proof We remarked in Property 5 that if (S,t) is optimum, any swapped schedule $(S_{(i)},t_{(i)})$ for $i = 1,\dots,h$ is optimum as well. This is true for any composition of swappings. Thus, the following sets contain the same elements: $\{A(i),B(i)\} = \{2i,2i-1\}$ for $i = 1,\dots,h$. We further know by Property 2 that in an optimum schedule, both A and B must be ordered nondecreasingly by base length. Hence, we can now deduce the job sequence S. The makespan $Q(S,t)$ then is obviously calculated by the formula in the proof of Property 5, thereby setting $t = M - Q_0(A,0)$. □

Note that, for $n = 2h+1$ jobs, there exist 2^h different optimum schedules, as for each of the h job pairs, a binary decision is to be made.

5 NP Completeness Proofs

We use the properties of the cfDTMP from section 4 to prove NP hardness of the cDTMP, and its generalization, the DTMP. Moreover, the fDTMP turns out to be NP hard, even with its flexible start time. However, the subproblem cfDTMP with a common ideal time turns out to be in P (see section 6).

Garey et al. (1988) used several properties of the early/tardy scheduling problem with a common due date to prove NP hardness of the general early/tardy problem in its decision version. The reduction uses a partition problem. In the corresponding scheduling instance, the solution divides the partition jobs into two halves around their common due date, thereby achieving an optimum V-shape. A further requirement is that a blocker job, which has a different due date, is neither early nor tardy. This ensures, for any No-instance, that the minimum objective is above threshold. Our proof follows a similar reduction of a partition problem. Meanwhile, the steps are different, due to the time dependent processing times. Moreover, our proof is not in need of blocker jobs. Therefore, it shows that even the cDTMP decision problem (with one common ideal time) is NP hard. For the fDTMP however, (only) one further blocker job with another ideal time is needed to show NP hardness.

In section 5.1, we introduce a modification of the Even Odd Partition Problem. This problem is used in section 5.2 to show the NP hardness of the decision versions of the cDTMP, the DTMP, and the fDTMP.

5.1 A Modified Even Odd Problem

From the well-known Even Odd Partition Problem (Garey et al., 1988), we define a variant. We then show that this modification is NP hard.

Definition 4 (Even Odd Partition Problem (Garey et al., 1988))

Instance: Given a set of $n = 2h$ positive integers $Y = \{y_1,\dots,y_n\}$ where $y_j < y_{j+1}$ for all $j = 1,\dots,n-1$.

Question: Is there a partition of Y into subsets Y_1 and $Y_2 := Y \setminus Y_1$ such that $\sum_{y\in Y_1} y = \sum_{y\in Y_2} y$, and such that for each $i = 1,\dots,h$, the set Y_1 contains exactly one element of set $\{y_{2i-1},y_{2i}\}$?

The modification entails the additional requirement that the given $n = 2h$ elements x_j, $j = 1,\dots,n$, are sorted increasingly even when divided by $\frac{1+b}{2}b^{h-\lfloor j/2 \rfloor}$ for a finite $b > 1$.

Definition 5 (Modified Even Odd Partition Problem)

Instance: Given a finite rational number $b > 1$ and a set of $n = 2h$ positive integers $X = \{x_1,\dots,x_n\}$. Let $B_0 = 0$ and $B_i = \frac{1+b}{2}b^{h-i}x_{2i}$ for all $i = 1,\dots,h-1$. Assume $x_{2i+1} > B_i$ and $x_{2i+2} > x_{2i+1}$ for all $i = 0,\dots,h-1$.

Question: Is there a partition of X into subsets X_1 and $X_2 := X \setminus X_1$ such that $\sum_{x\in X_1} x = \sum_{x\in X_2} x$, and such that for each $i = 1,\dots,h$, the set X_1 contains exactly one element of set $\{x_{2i-1}, x_{2i}\}$?

The division of a nondecreasing number series by a nonincreasing number series results in a nondecreasing number series. For $i = 1,\dots,n$, the given x_i's are ordered nondecreasingly, and the number series $\frac{1+b}{2}b^{h-\lfloor i/2 \rfloor}$ is ordered nonincreasingly. Hence, the x_i's are still ordered nondecreasingly even when divided by $\frac{1+b}{2}b^{h-\lfloor i/2 \rfloor}$.

We now construct a reduction of the Even Odd Partition Problem to the Modified Even Odd Partition Problem. Lee and Vairaktarakis (1993, p. 290) made a similar reduction for a different definition of B_i, namely $B_i = \sum_{k=1}^{i} x_{2k}$ for $i = 0,\dots,h-1$.

Theorem 1 The Modified Even Odd Partition Problem is NP hard.

Proof Given an instance of the Even Odd Partition Problem by the set $Y = \{y_1,\dots,y_n\}$. From Y, we iteratively construct an instance X of the Modified Even Odd Partition Problem as follows: For B_i defined as above, let $x_i = B_{\lfloor (i-1)/2 \rfloor} + y_i$ for $i = 1,\dots,n$. Then, for all $i = 0\dots h-1$ we have $x_{2i+2} > x_{2i+1}$ and $x_{2i+1} > B_i$ fulfilled. Thus, instance X satisfies the constraints of the Modified Even Odd Partition Problem. Also, this construction can be done in polynomial time.

Then, we need to show that instance Y is a Yes-instance if and only if X is. Suppose Y_1, Y_2 is a solution for Y. Let I_1, I_2 be the set of indices of the elements of Y_1, Y_2 respectively, ordered in increasing order. Then,

$$\begin{aligned}
&\sum_{i\in I_1} y_i = \sum_{i\in I_2} y_i \\
\iff &\left(\sum_{i\in I_1} B_{\lfloor (i-1)/2 \rfloor}\right) + \left(\sum_{i\in I_1} y_i\right) = \left(\sum_{i\in I_2} B_{\lfloor (i-1)/2 \rfloor}\right) + \left(\sum_{i\in I_2} y_i\right) \\
\iff &\sum_{i\in I_1} (B_{\lfloor (i-1)/2 \rfloor} + y_i) = \sum_{i\in I_2} (B_{\lfloor (i-1)/2 \rfloor} + y_i) \\
\iff &\sum_{i\in I_1} x_i = \sum_{i\in I_2} x_i.
\end{aligned}$$

Define $X_1 = \{x_i \mid i \in I_1\}$ and $X_2 = \{x_i \mid i \in I_2\}$. Then, $\sum_{x_i\in X_1} x_i = \sum_{x_i\in X_2} x_i$. As well, X_1 contains precisely one of $\{x_{2i-1}, x_{2i}\}$ because Y_1 contains precisely one of $\{y_{2i-1}, y_{2i}\}$ for all $i = 1,\dots,n$. Hence, the sets X_1, X_2 constitute a Yes-instance for X.

Conversely, if X_1, X_2 is a Yes-instance for X, the above equivalences mean that the sets $Y_1 = \{y_i \in Y \mid v_i \in X_1\}$ and $Y_2 = \{y_i \in Y \mid x_i \in X_2\}$ constitute a solution of Y. Therefore, X is a Yes-instance if and only if Y is. □

5.2 Problem Reduction

We show the NP hardness of the decision version of the DTMP and two variants, the cDTMP and the fDTMP. The properties from section 4 are a prerequisite. The proof is made by reducing the Modified Even Odd Partition Problem introduced in section 5.1. Please note that the latter problem is defined for rational numbers only, in order to be computable on a nondeterministic Turing machine (or an equivalent). The DTMP instead allows real numbers in the input, which is more general. However, as this formulation is a generalization, it still allows showing NP hardness by reduction.

Theorem 2 The decision versions of the cDTMP and the DTMP are NP hard.

Proof We begin with an instance of the Modified Even Odd Partition Problem (see Definition 5), given by a finite rational number $b > 1$ and a set of $n = 2h$ positive integers $X = \{x_1, \ldots, x_n\}$ obeying the required constraints.

Now, we can define an instance of the cDTMP:

- We know $b > 1$. Let $a := \frac{2b-2}{b+1}$, thus $a \in (0,2)$ as required. Solving a for b, we have $b = \frac{2+a}{2-a}$. Moreover, $\frac{1+b}{2} b^x = \frac{2}{2-a} b^x = \frac{2}{2-a} f(x) = g(x)$ (see Definition 2) for all $x \in \mathbb{Z}$.
- Then, we define the set J of $n+1$ jobs $J = \{0, \ldots, n\}$ with lengths $l_0 = 0$ and $l_j := x_j / g(h - \lfloor j/2 \rfloor)$ for $j = 1, \ldots, n$. Hereby, the jobs $0, \ldots, n$ are sorted by nondecreasing base length.
- We deduct from Property 7, that if we interpret a and J with an arbitrary common ideal midtime as a cfDTMP instance, then an optimum schedule of the jobs J results in a makespan of $q := l_0 f(h) + \sum_{i=1}^{h} (l_{2i} + l_{2i-1})\, g(h-j)$. Using q, we set a common ideal midtime $M_j = M = q/2$ for all jobs $j \in J$.
- Furthermore, we set the start time $t = 0$.

Finally, we ask whether there is, for the given problem, a schedule (S,t) where for the objective function $Q(S,t) = \sum_{j=0}^{n} p_j \leq q$ holds?

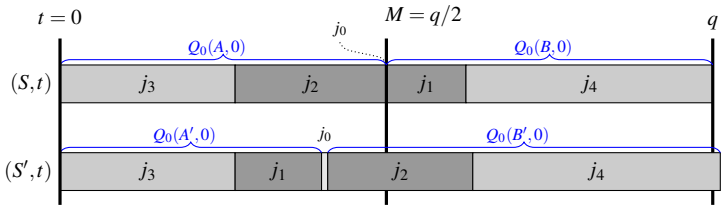

Fig. 9 The Modified Even Odd Partition Problem can be reduced to the cDTMP, and the generalized DTMP. In this example, with $n = 4$ elements, it corresponds to scheduling 5 jobs $j_0, \ldots, j_4$, where j_0 has $l_0 = 0$, such that $C_{\max} \leq k$. If the jobs are nondecreasingly sorted by base length, a minimum makespan schedule is achieved by placing j_0 in the middle, and alternately prepending and appending the following jobs. Now, $C_{\max} \leq k$ if and only if j_0 is at M, as in the visualized schedule (S,t).

The optimum makespan q can obviously only be reached if $p_0 = l_0 = 0$, implying $m_0 = M$. This happens if and only if the makespan can be split into two equal sized parts

$Q_0(A,0) = Q_0(B,0) = q/2 = M$ (see Figure 9), creating a partition of $J \setminus \{0\}$ into sets A,B:

$$\begin{aligned} & Q_0(A,0) = Q_0(B,0) \\ \iff & \sum_{i=1}^{h} l_{A(i)}\, g(h-i) = \sum_{i=1}^{h} l_{B(i)}\, g(h-i) \\ \iff & \sum_{i=1}^{h} x_{A(i)} = \sum_{i=1}^{h} x_{B(i)}. \end{aligned}$$

The last equation solves the Modified Even Odd Partition Problem instance, concluding the problem reduction.

Therefore, the decision versions of the cDTMP, and the generalized DTMP are NP hard. Furthermore, as, for a given (S,t) the question $Q(S,t) \leq k$ can be answered in polynomial time, they both are NP hard. □

Theorem 3 The decision version of the fDTMP is NP hard.

Proof From a Modified Even Odd Partition Problem instance, construct a fDTMP instance equivalent to the cDTMP instance in the proof of Theorem 2, except for the specification of t, which is a variable here. Add one more job with zero base length to J: job $n+1$ with ideal midtime $M_{n+1} = 0$. Then we ask the question whether there exists a schedule with $Q(S,t) \leq q$. Remember that the optimum makespan of the jobs in set $\{1,\dots,n\} \subset J$ is q. Therefore, the answer to the question is Yes, if and only if job $n+1$ can be placed at its ideal midtime, thus having zero processing time. This only happens when the job subset $\{1,\dots,n\}$ can be partitioned in sets A,B such that $Q_0(A,0) = Q_0(B,0) = q/2$. The reduction then continues like the proof of Theorem 2. Therefore, the decision version of the fDTMP is NP hard. We remark that even an idle time would be of no use. This could, e.g., be inserted between job $n+1$ and the others, in order to set $p_{n+1} = 0$. But as previously noted, for the given $|a| < 1$, any idle time only increases the makespan. □

6 Exact Polynomial Time Algorithm for the cfDTMP

While the cDTMP and the fDTMP are NP hard, the cfDTMP is solvable in polynomial time. We show this by presenting an exact polynomial time algorithm. It is later used in a lower bound for the DTMP (see section 7.4.8). As a side note, we remark that this problem is similar to the classic scheduling problem of minimizing early/tardiness with idle time and a nonrestrictive (i.e., large) common due date. This problem is studied by Kanet (1981), he developed a polynomial time algorithm for it.

Property 7 is defined for cfDTMP instances with odd numbers of jobs only. Hence, we need to specify optimum schedules for instances with even numbers of jobs.

Property 8 Given a cfDTMP instance with two jobs $\{1,2\}$. Then, any schedule (S,t) with $m_1 \leq M \leq m_2$ has a minimum makespan, namely $Q(S,t) = \frac{l_1+l_2}{1-a/2}$.

Proof Plan jobs $1,2$ such that their midtimes are $m_1 \leq M \leq m_2$. Now, job 1 is directly before job 2, meeting at some time $x = C_1 = t_2$. By (1), we calculate the start time of job 1 by solving equation $C_1(t_1) = x$ for t_1. For this, we use the symmetric behavior of p_j depending

on $|m_j - M|$. This introduces $t_1(x) := M - (C_1(M + (M - x)) - M) = 2M - C_1(2M - x)$. Then, the makespan for both jobs is

$$\begin{aligned} p_1 + p_2 &= C_2(x) - t_1(x) \\ &= C_2(x) - 2M + C_1(2M - x) \\ &= \left(x + \frac{l_2 + a(x-M)}{1 - a/2}\right) - 2M + \left(2M - x + \frac{l_1 + a(2M - x - M)}{1 - a/2}\right) \\ &= \left(x + \frac{l_2 + a(x-M)}{1 - a/2}\right) - \left(x + \frac{l_1 - a(x-M)}{1 - a/2}\right) \\ &= \frac{l_1 + l_2}{1 - a/2}. \end{aligned}$$

Thus, all positions satisfying $m_1 \leq M \leq m_2$ result in the same minimum makespan. □

Property 9 Given a cfDTMP with an even number $n = 2h$ of jobs $J = \{1, \ldots, n\}$, sorted nondecreasingly by base length. Then, a schedule (S,t) has a minimum makespan $Q(S,t)$ if $m_1 \leq M \leq m_2$ or $m_1 \geq M \geq m_2$, and for the resulting sets $\bar{A}, \bar{B}$ there is $|\bar{A}| = |\bar{B}|$ and $\{\bar{A}(i-1), \bar{B}(i-1)\} = \{2i, 2i-1\}$ for $i = 2, \ldots, h$.

Proof For the given instance, by Property 2, any optimum schedule must place the shortest jobs 1, 2 around M, with no other job in between. As of Property 8, the minimum makespan of those two jobs is achieved if $m_1 \leq M \leq m_2$.

Let (S,t) be a schedule for the given instance with $m_1 \leq M \leq m_2$ or $m_1 \geq M \geq m_2$ and no other job between job 1 and 2. By Property 8, we have $p_1 + p_2 = \frac{l_1 + l_2}{1 - a/2}$. Consider the resulting sets $\bar{A}$, $\bar{B}$. Here, J is partitioned into sets $\{1,2\}$, $\bar{A}$, and $\bar{B}$. With Property 1, we can express the makespan for (S,t) as

$$\begin{aligned} Q(S,t) &= Q_0(\bar{A}, x(p_1 + p_2)) + Q_0(\bar{B}, (1-x)(p_1 + p_2)) && (19) \\ &= Q_0(\bar{A}, 0) + Q_0(\bar{B}, 0) + x(p_1 + p_2) f(|\bar{A}|) + (1-x)(p_1 + p_2) f(|\bar{B}|) && (20) \\ &= Q_0(\bar{A}, 0) + Q_0(\bar{B}, 0) + \frac{l_1 + l_2}{1 - a/2} \left(x f(|\bar{A}|) + (1-x) f(|\bar{B}|)\right). && (21) \end{aligned}$$

We are interested in optimum $\bar{A}$, $\bar{B}$, and x that achieve a minimum $Q(S,t)$.

The optimum value of x depends on the cardinalities of $\bar{A}$ and $\bar{B}$. We differ three cases: (a) if $|\bar{A}| > |\bar{B}|$ then $x = 0$, (b) if $|\bar{A}| < |\bar{B}|$ then $x = 1$, and (c) if $|\bar{A}| = |\bar{B}|$ then $x \in [0,1]$. As $|\bar{A}| + |\bar{B}| = n - 2$, the last term in (21) is minimum if $|\bar{A}| = |\bar{B}|$, hence we prefer case (c). There, any $x \in [0,1]$ is optimum.

To find an optimum partition of $J \setminus \{1,2\}$ into sets $\bar{A}$, $\bar{B}$, we construct a new cfDTMP instance with jobs $J' = \{\chi, 3, 4, \ldots, n\} = \{\chi\} \cup \bar{A} \cup \bar{B}$ with a new job χ of base length $l_\chi = 0$. Let (S',t') be a minimum makespan schedule. This yields $\bar{A}', \bar{B}'$. By Property 7, $\{\bar{A}'(i-1), \bar{B}'(i-1)\} = \{2i, 2i-1\}$ for $i = 2, \ldots, h$ and the makespan is

$$Q(S',t') = Q_0(\bar{A}', 0) + Q_0(\bar{B}', 0). \quad (22)$$

Sets $\bar{A}'$, $\bar{B}'$ are a valid partition of $J \setminus \{1,2\}$, and therefore constitute feasible values for $\bar{A}$, $\bar{B}$. By definition of (S',t'), they minimize (22). By letting $\bar{A} = \bar{A}'$, $\bar{B} = \bar{B}'$, (22) is a part of (21). Also, by Property 4, we have $|\bar{A}'| = |\bar{B}'|$, which is the prerequisite for the preferred case (c). Therefore, (21) is minimum for $\bar{A} = \bar{A}'$, $\bar{B} = \bar{B}'$, and $x \in [0,1]$. □

This generic property about optimum cfDTMP instances with even numbers of jobs enables us to state a polynomial time exact cfDTMP algorithm.

Algorithm 1 Given a cfDTMP instance with n jobs. Sort the jobs nondecreasingly by base length. Assume the resulting job list is $1,\dots,n$ for even n, or $0,1,\dots,n-1$ for odd n. Let $h := \lfloor n/2 \rfloor$. To construct an optimum job sequence S, we predetermine whether assigning a job to A or B: For $i = 1,\dots,\lfloor n/2 \rfloor$, assign job $2i$ to $A(i)$ and job $2i+1$ to $B(i)$. Accordingly, set start time $t = M - Q_0(\{2i \mid i = 1,\dots,\lfloor n/2 \rfloor\}, l_\chi/2)$ where $l_\chi := 0$ if n is even, or $l_\chi := l_0$ if n is odd. Then, the sequence S is $A(\lfloor n/2 \rfloor),\dots,A(1),B(1),\dots,B(\lfloor n/2 \rfloor)$ for even n, and $A(\lfloor n/2 \rfloor),\dots,A(1),\chi,B(1),\dots,B(\lfloor n/2 \rfloor)$ with $\chi = 0$ for odd n.

Such a schedule (S,t) satisfies either Property 7 if n is odd or Property 9 if n is even. Therefore, (S,t) is optimum. In the algorithm, sorting is the most expensive step, taking $O(n \log n)$ time. The remaining computations take $O(n)$ time. Concluding, the stated algorithm solves cfDTMP instances exactly and runs in polynomial $O(n \log n)$ time.

7 Algorithms for the DTMP

A common way to exactly solve new optimization problems is to use general purpose solvers for, e.g., mixed integer programming (MIP). Hence, we present an exact MIP formulation for the DTMP, which furthermore extends easily to the fDTMP (section 7.1). As it turns out, in our experiments, it is not fast enough even for moderate numbers of jobs n. Therefore, we present a dynamic programming algorithm (DP, see section 7.2), and simple heuristics (section 7.3). Furthermore, we deduce lower bounds on the makespan (section 7.4), which allow for a branch and bound algorithm (B&B, see section 7.5). For approximate solutions, the branch and bound search is truncated (TrB&B, see section 7.6).

7.1 Mixed Integer Programming

We state the DTMP as a mixed integer program (MIP). Given parameters are t, a, l_j, M_j for jobs $j \in J$. To encode the permutation of the n jobs $J = \{1,\dots,n\}$, we assign every job to a position $P = \{1,\dots,n\}$ in the sequence. The sequence $S : J \mapsto P$ then defines schedule (S,t). We offer n positions, each of which holds one job. Here, binary variables x_{jk} decide if job $j \in J$ is placed at position $k \in P$. For each position $k \in P$, we calculate a processing time $p_{[k]}$ and a completion time $C_{[k]}$. The objective to be minimized is $C_{[n]} - t$.

$$\min C_{[n]} - t \quad \text{subject to:}$$

$$t = C_{[0]}, \tag{23}$$

$$C_{[k-1]} + p_{[k]} = C_{[k]} \qquad \text{for all } k \in P, \tag{24}$$

$$\sum_{j \in J} l_j x_{jk} + a \left| C_{[k-1]} + \frac{p_{[k]}}{2} - \sum_{j \in J} M_j x_{jk} \right| = p_{[k]} \qquad \text{for all } k \in P, \tag{25}$$

$$\sum_{j \in J} x_{jk} = 1 \qquad \text{for all } k \in P, \tag{26}$$

$$\sum_{k \in P} x_{jk} = 1 \qquad \text{for all } j \in J, \tag{27}$$

$$x_{jk} \in \{0,1\} \qquad \text{for all } j \in J, k \in P. \tag{28}$$

Lines (23) and (24) recursively set $C_{[k]}$, $k \in P$. The processing time $p_{[k]}$ of the job at position $k \in P$ is expressed in line (25), closely following Definition 1. Note that the stated absolute value function can be linearized using auxiliary variables. Lines (26), (27), and (28) ensure that each job is assigned to exactly one position.

This MIP formulation contains no relaxations, hence it solves DTMP instances exactly. Our method of positional assignment can be categorized under "assignment and positional date variables" as in Keha et al. (2009).

Note that the formulation can easily be adopted for the fDTMP by changing parameter start time t to a variable.

7.2 Dynamic Programming

Dynamic Programming is a well known principle to incorporate recursive substructures for avoiding repeated computations. It is long since used in standard (fixed processing time) scheduling, e.g., beginning with Held and Karp (1962). Interestingly, it is as well applicable for our time dependent processing time problem. A necessary condition is the fact that the function $C_j(t)$ (specified in Definition 1) is monotonically increasing, as visualized in Figure 10. For this, first we show how the DTMP can be formulated as a recurrence relation.

Property 10 Given a DTMP with a job set J and a start time t. Now, $C^*_{\max}(J) = C_{\max}(S,t)$ of a minimum schedule (S,t) is

$$C^*_{\max}(J) = \begin{cases} t & \text{if } J = \emptyset, \\ \min\limits_{j \in J} \; C_j(C^*_{\max}(J \setminus \{j\})) & \text{else.} \end{cases}$$

Proof In any optimum schedule of the jobs J, some job $j \in J$ is the last job. Say it starts at t'. Because $C_j(t')$ is monotonically increasing, a smaller value for t' reduces the completion time. The jobs in the set $J \setminus \{j\}$ are to be scheduled before j. Thus, the shortest schedule for this set, completing at $t' = C^*_{\max}(J \setminus \{j\})$, also minimizes the completion time for j.

What remains is to find out which job j comes last. For this, we try any job $j \in J$ and take the one with smallest $C_j(C^*_{\max}(J \setminus \{j\}))$. Concluding, there is no other smaller value for $C^*_{\max}(J)$, and the equation is valid. □

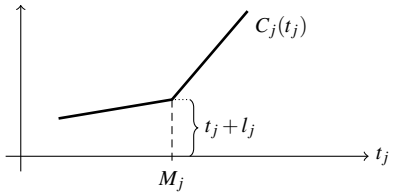

Fig. 10 Irrespective of factor $a \in (0,2)$, the completion time $C_j(t_j)$ of a job j (see (1)) is a monotonically increasing function of the job's start time t_j. The same monotonicity holds for a set of multiple jobs.

This recurrence relation can then be used in a dynamic programming algorithm, e.g., the one by Held and Karp (1962). This computes the value of the recurrence equation for all

subsets of J. For n jobs, the number of subsets is 2^n. As the number of operations is at most n for each subset, this algorithm runs in $O(n2^n)$ time.

7.3 Simple Heuristics

To quickly obtain a good solution for a given DTMP instance, we devise several simple, quickly implementable, but effective heuristics. Our heuristics in this section all begin with an initial solution that is improved with a local search.

The first heuristic sorts the jobs nondecreasingly by their ideal midtime M_j, and, as a tie-breaker, nondecreasingly by their base length l_j. Thereby, jobs should already lie close to their ideal midtimes. However, by Property 2, jobs with the same M_j value should form a V-shape. In this initial solution instead, they are only positioned well in the case of all being late. Therefore, this schedule is improved locally by a steepest ascent hill climbing search. The neighborhood consists of all swappings of two jobs. The best swap candidate is applied until we reach a local optimum. Together with the initial schedule, we call this the sorted local search heuristic (SLS).

The second heuristic constructs a V-shaped arrangement of jobs with the same ideal midtime M_j. For this, the heuristic forms job groups of common M_j values. The groups are appended in nondecreasing order of their M_j value. Each job group is arranged in a V-shape. A V-shape consists of two parts, the front and the rear part. A random value $u_j \in \{0,1\}$ for each job $j \in J$ decides whether job j shall be placed in the front or the rear part. All jobs belonging to the front part of a V-shape group are sorted nonincreasingly, while those in the rear part are sorted nondecreasingly. In this order, both parts are appended to the schedule. The resulting initial schedule then is improved by the same steepest ascent hill climbing search as described above. As the heuristic has random variables, we repeatedly run it n_{runs} times, and select the best of all runs. We call this the V-shape sorted local search heuristic (VLS).

The third heuristic starts with an arbitrarily permutated schedule. In contrast to the previous heuristics, this initial solution imposes an unbiased assignment of jobs to positions. I.e., there is no sorting by M_j values as before. This increases the diversity of the initial schedule. Therefore, we begin with a number of n_{runs} initial solutions. On each, a steepest ascent hill climbing search is applied. At the end, the best resulting schedule is selected. We call this the randomized local search heuristic (RLS).

7.4 Lower Bounds

Problem specific lower bounds are a necessity for developing an effective branch and bound algorithm. In this section, we describe several lower bounds for the DTMP, and summarize them into a single lower bound. For this, we make use of Property 2, that concerns with the optimality of V-shapes. Furthermore, we use the polynomial time algorithm for the cfDTMP (Algorithm 1) to derive a lower bound for a subset of jobs with a common ideal time. Note that, by NP hardness of the fDTMP (Theorem 3), a similar lower bound would be hard to compute for jobs of distinct ideal times. However, in certain cases, it is still possible to express bounds for job sets with distinct ideal times (see sections 7.4.6 and 7.4.7).

7.4.1 A Trivial Lower Bound (LB_0)

Given a set of jobs J and a start time t. By definition, $p_j \geq l_j$, $j \in J$. Therefore, a trivial lower bound for the makespan $Q(S,t) = C_{\max}(S,t) - t$ is $LB_0(J) := \sum_{i \in J} l_i$, This bound is tight if and only if there exists a schedule where $m_j = M_j$ for all $j \in J$.

7.4.2 Jobs with a Common Ideal Midtime Before Start Time (LB_{1a})

If we have a subset of jobs with a common ideal midtime $M \in \mathbb{R}$, we may improve LB_0. For this, we look at our given set of n jobs J where the ideal midtimes may differ. For a given start time t and some M, define subset $B_{t,M} := \{j \in J \mid M_j = M \wedge M_j - l_j/2 \leq t\}$. Then, $B_{t,M}$ contains those jobs that have M in common and, given the start time, must be scheduled after their ideal midtime, i.e. $m_j \geq M_j$ for $j \in B_{t,M}$. Without loss of generality, assume $B_{t,M} = \{1,\ldots,k\}$, sorted by nondecreasing base length. Hence $J \setminus B_{t,M} = \{k+1,\ldots,n\}$. Let $S_{B_{t,M}}(i) = i$ for $i = \{1,\ldots,k\}$. Then, define $LB_{1a}(J,t,M) := Q(S_{B_{t,M}}, t-M) + LB_0(J \setminus B_{t,M})$. By Property 1,

$$LB_{1a}(J,t,M) = (t-M)\,(f(k)-1) + \sum_{i=1}^{k} l_i\, g(k-i) + \sum_{i=k+1}^{n} l_i.$$

This represents the set of jobs scheduled after t. As required by Property 2, it is sorted nondecreasingly by base length. If no other jobs interfere, i.e. $B_{t,M} = J$, this bound is tight. On the other hand, if $B_{t,M} = \emptyset$, it degenerates to $LB_{1a}(J,t,M) = LB_0(J)$.

Because $B_{t,M} \cap B_{t,M'} = \emptyset$ for $M, M' \in \mathbb{R}$, $M \neq M'$, we can apply this lower bound separately to other ideal midtimes, and then combine their values. For this, we partition J into subsets of jobs with a common ideal midtime. This results in

$$LB_{1a}(J,t) := \sum_{M \in \{M_j \mid j \in J\}} LB_{1a}(\{j \in J \mid M_j = M\}, t, M).$$

Note that $LB_{1a}(J,t) \geq LB_0(J)$. Also, $LB_{1a}(J,t) = LB_0(J)$ if and only if there is no job $j \in J$ with $M_j - l_j/2 < t$.

7.4.3 Jobs with a Common Ideal Midtime After Upper Bound (LB_{2a})

We can apply LB_{1a} symmetrically to jobs at the end of the schedule. Given a job set J and an ideal midtime M. Denote the upper bound value UB of $C_{\max}(S,t)$ by C. Define job set $A_{C,M} = \{j \in J \mid M_j = M \wedge M_j + l_j/2 \geq C\}$. As C is an upper bound on $C_{\max}(S,t)$, any job $j \in A_{C,M}$ must be scheduled both before C and its ideal midtime, i.e. $m_j \leq M$. Therefore, we can apply LB_{1a} symmetrically and define

$$LB_{2a}(J,C,M) := LB_{1a}(A_{C,M}, M + (M-C)) + LB_0(J \setminus A_{C,M}).$$

This bound is tight if, given a start time t, we have $C - t = LB_{2a}(J,C,M)$ and $A_{C,M} = J$. On the other hand, if $A_{C,M} = \emptyset$, then $LB_{2a}(J,C,M) = LB_0(J)$.

To apply this bound for all ideal midtimes, define

$$LB_{2a}(J,C) := \sum_{M \in \{M_j \mid j \in J\}} LB_{2a}(\{j \in J \mid M_j = M\}, C, M).$$

Note that $LB_{2a}(J,C) \geq LB_0(J)$. Also, $LB_{2a}(J,t) = LB_0(J)$ if and only if there is no job $j \in J$ with $M_j + l_j/2 > C$.

7.4.4 Jobs with Different Ideal Midtimes Before Start Time (LB_{1b})

Given a start time t, and a set of n jobs J. Define set $B_t = \{j \in J \mid M_j - l_j/2 \leq t\}$. Note that $B_t = \bigcup_{M \in \mathbb{R}} B_{t,M}$. At t, the processing time of any job $j \in B_t$ has increased to $l_j + a(t - M_j)$, respectively. Hence, we let $l'_j := l_j + a|t - M_j|$ and $M'_j = t$ for all $j \in B_t$. We denote this transformed job set by $\tau(B_t, t)$. Now, t is the virtual common ideal midtime for the jobs in $\tau(B_t, t)$. Therefore, we can apply LB_{1a}. This defines

$$LB_{1b}(J,t) := LB_{1a}(\tau(B_t,t),t) + LB_0(J \setminus B_t).$$

This bound is tight if $B_t = J$. Also note that $LB_{1b}(J,t) \geq LB_{1a}(J,t)$.

7.4.5 Applying LB1 to Jobs After Upper Bound (LB_{2b})

We can similarly define LB_{2b}. Given an upper bound value C and a job set J. Define set $A_C = \{j \in J \mid M_j + l_j/2 \geq C\}$. The transformed set again is $\tau(A_C, C)$. Now, we can define

$$LB_{2b}(J,C) := LB_{2a}(\tau(A_C,C),C) + LB_0(J \setminus A_C).$$

Note $LB_{2b}(J,C) \geq LB_{2a}(J,C)$. This bound is tight again if, given a start time t, we have $C - t = LB_{2b}(J,C)$.

7.4.6 Enhancing LB_{1b} (LB_{1c})

Given a start time t and a job set $J = \{1,\ldots,n\}$ sorted by nondecreasing $(l_j - aM_j)$. Remember from LB_{1b} that we applied the transformation τ only to the jobs in set B_t. Those jobs, when started at t, complete earliest at some $t' > t$. Reusing t' as a new start time, a different set of jobs $B_{t'}$ emerges. The larger t' is, the more jobs are in set $B_{t'}$, hence $B_{t'} \supseteq B_t$ holds. Observe that we now are allowed to append jobs $\{j \in \tau(B_{t'}, t') \mid j > \max\{i \mid i \in \tau(B_t, t')\}\}$, i.e. those jobs of $B_{t'}$ that at t' have a higher base length than all previously appended jobs B_t.

This consideration can be formalized as follows. Define

$$r(J,t,j) := \begin{cases} 0 & \text{if } j \notin J, \\ (C_j(t) - t) + r(J, C_j(t), j+1) & \text{if } M_j - l_j/2 \leq t, \\ l_j + r(J,t,j+1) & \text{else.} \end{cases}$$

Then, $LB_{1c}(J,t) := r(J,t,1)$. Note $LB_{1c}(J,t) \geq LB_{1b}$. Furthermore, this bound is tight if the third case in r never triggers.

7.4.7 Enhancing LB_{2b} (LB_{2c})

This symmetrically applies to the end of our schedule. Given an upper bound value C and a job set $J = \{1,\ldots,n\}$ sorted by nondecreasing $(l_j + aM_j)$. First, define $t_j(C) := M_j - (C_j(M_j + (M_j - C)) - M_j) = 2M_j - C_j(2M_j - C)$, which corresponds to the start time of a job $j \in J$ that completes at C. Define

$$s(J,C,j) := \begin{cases} 0 & \text{if } j \notin J, \\ (C - t_j(C)) + r(J, t_j(C), j+1) & \text{if } M_j + l_j/2 \geq t, \\ l_j + r(J,C,j+1) & \text{else.} \end{cases}$$

Then, $LB_{2c}(J,C) := s(J,C,1)$. Note $LB_{2c}(J,t) \geq LB_{2b}$. If the third case in s never triggers and if, given a start time t, we have $C - t = LB_{2c}(J,C)$, then the bound is tight.

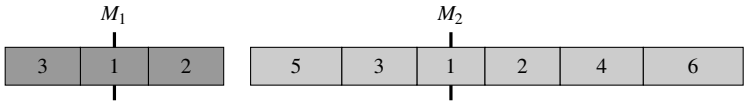

Fig. 11 LB_3 groups jobs of same ideal midtime into locally optimum V-shapes.

7.4.8 Lower Bound on Common Ideal Midtime Job Sets (LB_3)

Consider a subset of jobs $J_M \subseteq J$ with a common ideal midtime $M \in \mathbb{R}$: $J_M := \{j \in J \mid M_j = M\}$. Obviously, not every job $j \in J_M$ can be scheduled such that $m_j = M$. But we can efficiently compute the minimum cDTMP makespan of this set, denoted by q. To obtain q, we run the $O(n \log n)$ time Algorithm 1 for the cDTMP (see section 6). Then, q delivers a lower bound for J_M, denoted by $LB_3(J_M, M) := q$.

For problems with different ideal midtimes, the job set J is partitioned into subsets of common ideal midtime to compute individual lower bounds:

$$LB_3(J) = \sum_{M \in \{M_j \mid j \in J\}} LB_3(J_M, M).$$

This is also depicted in Figure 11.

7.4.9 Joint Lower Bound

Given a DTMP instance with a job set J, a start time t, and an upper bound value $C = UB$. We may summarize the presented bounds into a single lower bound.

First, we assemble job subsets $J_A, J_B \subseteq J$ for which LB_{1c} and LB_{2c} hit. For this, find the maximum cardinality subset $J_A \subseteq J$ where for all jobs $j \in J_A$, there is $LB_{1c}(J_A, t) > LB_{1c}(J_A \setminus \{j\}, t) + l_j$. Similarly, find the maximum cardinality subset $J_B \subseteq J$ where for all jobs $j \in J_B$, there is $LB_{2c}(J_B, t) > LB_{2c}(J_B \setminus \{j\}, t) + l_j$. Obviously, finding a largest subset is easy. It is equal to excluding jobs $j \in J$ that hit the third case in r and s when calculating $LB_{1c}(J, t)$ and $LB_{2c}(J, t)$, respectively.

Let $J_C = J \setminus (J_A \cup J_B)$. Then, define the lower bound for the makespan $Q(S, t)$ as

$$LB(J, t) := LB_{1c}(J_A, t) + LB_{2c}(J_B, t) + \sum_{M \in \{M_j \mid j \in J_C\}} LB_3(\{j \in J_C \mid M_j = M\}).$$

7.5 Branch and Bound Algorithm

In this section, we present a branch and bound algorithm B&B for solving the DTMP exactly. This uses the lower bound from section 7.4. To initialize the upper bound, it runs heuristics from section 7.3. To further speed up the search, two dominance rules are introduced in section 7.5.1. In section 7.5.2, our branching strategy is explained.

We base the B&B on a recursive depth first search, and plan jobs forwardly. A branch node is a pair (J_O, t'), consisting of a set of open jobs J_O and a partial completion time t'. We begin with the branch node $(J, 0)$ (assuming start time $t = 0$). From each branch node, we recurse to any job $j \in J_O$, thereby removing j from J_O and increasing t' by p_j.

As the currently best schedule, we use the best solution of both the LS and the RLS (see section 7.3). This initializes the makespan upper bound UB. To estimate the quality

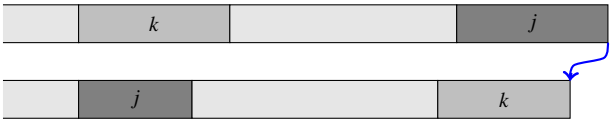

Fig. 12 Eliminate a branch if swapping any previous job k with the last job j lowers the partial completion time.

of a branch node, we use the lower bound described in section 7.4. In this way, a branch node (J_O,t') can be eliminated if $t'+LB(J_O,t') \geq UB$. When reaching $(\emptyset,t')$ for some t', we have a new best schedule, and UB can be lowered to t'. After having recursed to all $j \in J_O$ in all branch nodes, the best schedule represents an optimum solution for the problem instance.

7.5.1 Dominance Rules

In the following, we present two dominance rules for the DTMP, namely DR_1 and DR_2. When a dominance rule observes that a branch certainly leads to no improved solution, we eliminate the branch.

The dominance rule DR_1 checks if in the current branch node (J_O,t'), a shorter makespan has previously been found for the subset of completed jobs $J_C := J \setminus J_O$. By Property 10, the shorter partial solution is dominant. When reaching a certain J_C set for the first time, we store the upper bound of t' for this subset, $t'_{UB}(J_C) := t'$, and continue. When we branch to the same J_C set again with another job sequence, we compare this sequence's t' with $t'_{UB}(J_C)$. If $t' \geq t'_{UB}(J_C)$, we eliminate the branch. Else, we update $t'_{UB}(J_C)$. The number of J_C sets is up to 2^n. Thus, the memory requirements for this dominance rule can be very high. Therefore, it is useful to only store a limited number of t'_{UB} values. In doing so, the number of stored values may grow to that limit. Then, some values need to be discarded, e.g., the least recently used values.

The dominance rule DR_2 checks if the current partial solution can be improved by performing a quick neighborhood search. In our upper bound heuristic, we use a job swapping neighborhood. Here, we use a similar neighborhood which can be described as follows. In the current branch node, the job set J_C completes at t'. During branching, we recursively append jobs. Hence, we only look at the neighborhood which includes a change of the position of the last job j in the partial sequence. We allow swapping job j with a preceding job k, where additionally $S(k) \geq \max\{1, n-S(j)\}$ to reduce the number of checks later in the search tree. If for some k, the swap results in a partial completion time t'' with $t'' < t'$, we eliminate the branch. This is exemplified by Figure 12.

7.5.2 Branching Strategy

In order to fathom many branches quickly, we try to obtain good upper bounds for job subsets, namely the $t'_{UB}(J_C)$ values (see section 7.5.1), as early as possible. For this, the order of visiting subbranches is relevant. In this section, we explain the branching strategy which we develop for our B&B.

We remember that good solutions often exhibit V-shaped job orders. Hence, we prefer branches that actively establish V-shapes.

Given a branch node (J_O,t'). Consider jobs which are already after their ideal midtime: the set $B_{t'} = \{j \in J_O \mid M_j - l_j/2 \leq t'\}$. Secondly consider those jobs which are before or at their ideal midtime: the remaining set $J_O \setminus B_{t'}$, denoted by $A_{t'}$.

Remember that if $B_{t'} = J_O$, the optimum arrangement of set $B_{t'}$ sorts jobs nondecreasingly by $l_j - aM_j$. In the case of $B_t \subsetneq J_O$, we can sort the rest of the jobs inversely; that means nondecreasingly by $l_j + a(M_j - t') = C_j(t')$.

This leads to the following branching strategy:

1. Branch to jobs $j \in B_{t'}$, ordered nondecreasingly by the value of $C_j(t')$,
2. Branch to jobs $j \in A_{t'}$, ordered nonincreasingly by the value of $C_j(t')$.

It is costly to sort the sets $A_{t'}, B_{t'}$ in each recursion step. But it is possible to precompute both sets and their respective order for all distinct values of t' in polynomial time. This uses the fact that the sets and their order remain equal for all $t' \in (M, M']$ in an interval between consecutive ideal midtimes M and M'.

When computing $LB(J_O, t')$ (see section 7.4.9), it may occur that its J_B set equals J_O, and furthermore $LB_{2c}(J_B, t')$ is tight. Then, we already know the optimum sequence of J_O. In this case, it is always obtained by following the first subbranch.

Concluding, we first identify the interval q which contains t'. This yields the precomputed partition of J into A^q, B^q. Then, we branch to jobs in order of the sorted set $B^q \cap J_O$, then $A^q \cap J_O$. Only if $J_B = J_O$ and $LB_{2c}(J_B)(J_B, t')$ is tight, omit all except the first subbranch.

7.6 Truncated Branch and Bound Heuristic

The B&B (section 7.5) delivers exact solutions for the DTMP. We further apply its principles to construct an efficient, dedicated heuristic. For this, the search tree is truncated. This is the foundation of our truncated branch and bound heuristic TrB&B.

In the B&B, in each branch node, children branch nodes are discarded through bounding or dominance rules. The TrB&B discards even more children branch nodes. Particularly, only the n_{expl} children branch nodes (J_O, t') with the lowest $LB(J_O, t')$ values are explored; the others are discarded. We choose the value for n_{expl} depending on the level in the search tree. We explore more options at the beginning than at the end of the search. To achieve this, we factor in the number of open (i.e. not completed) jobs, $|J_O|$, as defined in Sec. 7.5. Note that $|J_O|$ is large at the beginning of the search, and small at the end. To explore more than ψ children at all times, we set $n_{\text{expl}} := \max\{\psi, \lfloor |J_O|/\sigma \rfloor\}$, for some integer $\psi, \sigma > 0$.

8 Numerical Results

8.1 Instance Generation

We generate a test set for the DTMP that allows for algorithm performance comparisons. It covers a variety of different problem settings to check for strengths and weaknesses of algorithms.

Generally, we set start time $t = 0$. The instances have a size of $n = 20, 24, \ldots, 60$ jobs. We used four variants $b = 1, \ldots, 4$ of generating a job's base length:

Case $b = 1$: base lengths $l_1, \ldots, l_n = 1$, all equal.
Case $b = 2$: base lengths $l_1 = 1, l_2 = 2, \ldots, l_n = n$, all distinct.
Case $b = 3$: lengths are drawn uniformly from $\{1, 2, \ldots, 10\}$.
Case $b = 4$: lengths are drawn from the geometric random variable $X = \lceil -\lambda \ln U \rceil$ where U is uniformly distributed in $[0, 1]$, and $\lambda = 2$.

The jobs are randomly assigned to ideal midtimes. For this, we define $|M| = n/4$ uniformly distributed $[0,1]$ random values as preparatory ideal midtimes. Then, each job is randomly assigned to one of these values. We now have $11 \cdot 4 = 44$ configurations, for each we generate 30 samples. Each sample yields 5 instances by varying a as follows.

For an instance, we further need the growth factor a and the ideal midtimes. For this, we vary $a \in \{0.05, 0.1, 0.2, 0.4, 0.6\}$. According to a, we finally obtain the ideal midtimes by scaling the preparatory ideal midtimes to interval $[M_{\min}, M_{\max}]$ being defined below. Thereby, we allow for direct performance comparisons on different values of a. The interval is calculated as follows. For an instance with jobs J of common ideal midtime $M = t = 0$, the makespan lower bound $LB_{1a}(J,0)$ (see section 7.4.2) is tight. Here, $LB_{1a}(J,0) = \sum_{i=1}^{n} l_i\, g(n-i)$ for the job set $J = \{1,\ldots,n\}$, sorted nondecreasingly by base length. Hence, the optimum $C^*_{\max} = LB_{1a}(J,0)$ of this instance can be calculated in polynomial time. We note that an arbitrary instance with some $M < 0$ is equivalent to an instance with $M' = 0$, where each job's base length is increased by $-aM$. To avoid such duplicate instances in the test set, we set $M_{\min} = 0$. A very large value of M exhibits the same behavior symmetrically. There, the optimum sequence sets the jobs nonincreasingly by base length. In fact, this job sequence is optimum for all instances with $M \geq \sum_{i=1}^{n} l_i\, g(n-i)$. Thus, we let $M_{\max} = \sum_{i=1}^{n} l_i\, g(n-i)$.

For each of the $44 \cdot 30$ samples, we have 5 values of a, yielding 6600 test instances.

8.2 Experimental Setup

For the given testbed, we obtain optimum schedules using three algorithms: MIP (section 7.1), DP (described in section 7.2), and B&B (section 7.5). For comparison, we evaluate the heuristics LS, RLS (both in section 7.3), and TrB&B (section 7.6).

We implemented our algorithms in C++, compiled with GCC 4.8.1. For the MIP formulation, we used the Gurobi 5.6.1 solver. To ensure precise and repeatable measurements, we ran the experiments on a headless dedicated machine, equipped with an Intel Core 2 Q9550 CPU, 4 GB RAM, operating a 64 bit Ubuntu Server Linux 13.10 and Kernel 3.11.0. Each instance was executed separately, allowing a maximum runtime of 480 seconds. To allow for a pure runtime measurement, we refrain from parallel computations.

As parameters for the VLS and the RLS we use $n_{\text{runs}} = n/4$. For the TrB&B, we use $\psi = 7$ and $\sigma \in \{3,6\}$. In the B&B we initialize the upper bound with the minimum value of SLS and RLS ($n_{\text{runs}} = n/4$).

8.3 Results and Discussion

To compare our solution methods for the DTMP, we first look upon the results for the MIP and the DP. Then we analyze the results for the other algorithms more in-depth.

The MIP solver completes the computations within a median of 31.36 seconds for the smallest instances of size $n = 20$. For $n = 24$, more than half of all instances already run over the time limit of 480 seconds. The same happens for $n = 28$, therefore we omit a test of the MIP for instance sizes larger than $n = 28$.

The results for the DP are more promising, at least for small instances. The median runtimes are 0.13 seconds for $n = 20$, 2.73 for $n = 24$, and 54.60 for $n = 28$. The DP runtime for a given instance size is quite consistent and predictable. This is supported by the fact that the coefficient of variation (ratio of the standard deviation to the mean) of the runtime is quite

n	B&B	SLS	VLS	RLS	TrB&B for $\sigma = 6$	TrB&B for $\sigma = 3$
20	100.00%	78.17%	89.17%	92.83%	97.67%	97.00%
24	100.00%	71.67%	82.17%	86.17%	94.67%	94.33%
28	100.00%	64.83%	74.50%	80.17%	92.83%	91.83%
32	100.00%	59.50%	68.17%	74.00%	86.33%	86.83%
36	99.83%	49.08%	58.93%	65.28%	81.47%	81.97%
40	98.17%	43.29%	51.78%	55.86%	75.55%	79.12%
44	93.00%	38.35%	46.95%	52.51%	73.66%	76.52%
48	82.50%	37.37%	45.45%	52.73%	73.94%	81.62%
52	79.00%	34.18%	43.25%	44.73%	70.89%	80.38%
56	66.00%	31.57%	37.37%	37.63%	69.70%	83.33%
60	54.33%	30.98%	35.28%	38.65%	76.07%	84.66%
all	88.44%	51.07%	59.91%	64.42%	82.22%	85.57%

Table 1 This table presents the fraction of instances solved to optimality by the exact B&B algorithm. For each heuristic, this table presents the fraction of optimally solved instances (of those where the optimum is known). Each row displays the results for the subset of instances with a specific number of jobs n. The bottom row shows the results of all instances together.

n	B&B	SLS	VLS	RLS	TrB&B for $\sigma = 6$	TrB&B for $\sigma = 3$
20	0.00	0.00	0.00	0.00	0.00	0.00
24	0.01	0.00	0.00	0.01	0.00	0.00
28	0.02	0.00	0.01	0.03	0.01	0.01
32	0.06	0.00	0.03	0.06	0.03	0.04
36	0.26	0.00	0.06	0.12	0.07	0.16
40	1.00	0.00	0.10	0.20	0.12	0.49
44	2.56	0.01	0.16	0.36	0.40	1.49
48	10.54	0.02	0.24	0.57	0.78	5.15
52	22.91	0.02	0.35	0.98	2.26	13.91
56	91.56	0.04	0.56	1.52	4.68	42.44
60	239.94	0.05	0.83	2.26	6.08	135.72
all	0.36	0.00	0.07	0.20	0.09	0.22

Table 2 This table displays algorithm performance by number of jobs n. The bottom row shows the results of all instances together. Displayed for each algorithm is the median runtime in seconds (including the runtime of those over the time limit).

low: 0.080 for $n = 20$, 0.084 for $n = 24$, and 0.087 for $n = 28$. The reason for this consistent runtime for same values of n is that any such instance runs through the same number of iterations. All of the instances with $n = 32$, however, run over the time limit, therefore we omit to test the DP with instances larger than that. It is visible that neither the MIP, nor the DP are able to solve larger instances. A reason is that none of them utilizes underlying, specific properties of the DTMP.

The B&B algorithm (see section 7.5) is able to solve larger instance sizes. For the following results, we refer to Table 1. The B&B obtains exact solutions for 88.44% of all instances. For $n \leq 32$, all instances are solved to optimality within the time limit. Hence it is able to supersede the other exact algorithms completely. Even for larger numbers of jobs n up to $n = 60$, it is able to solve most instances. Table 2 shows the median runtime of the B&B. The exponential growth of the runtime in relation to n is clearly visible. The number

n	SLS	VLS	RLS	TrB&B for $\sigma = 6$	TrB&B for $\sigma = 3$
20	0.07%	0.04%	0.04%	0.01%	0.01%
24	0.12%	0.07%	0.10%	0.01%	0.02%
28	0.14%	0.10%	0.12%	0.03%	0.03%
32	0.17%	0.13%	0.12%	0.05%	0.05%
36	0.21%	0.19%	0.17%	0.07%	0.05%
40	0.25%	0.21%	0.23%	0.08%	0.06%
44	0.29%	0.23%	0.28%	0.10%	0.09%
48	0.26%	0.23%	0.48%	0.12%	0.08%
52	0.27%	0.24%	1.15%	0.11%	0.08%
56	0.28%	0.25%	1.48%	0.09%	0.04%
60	0.28%	0.26%	2.91%	0.06%	0.04%
all	0.20%	0.17%	0.50%	0.06%	0.05%

Table 3 For each heuristic displayed is the mean absolute percentage deviation of the optimum value (if the optimum is known), by number of jobs n. The bottom row shows the results of all instances together.

of solved instances decreases exponentially as well. This is a behavior we expect from a NP hard problem setting.

The heuristics we compare are the SLS, the VLS, and the RLS from section 7.3, as well as the TrB&B from section 7.6. The SLS is fastest by a large margin. Its runtime is, in most cases, even too small to measure (with a 10 millisecond precision). Its maximum runtime over all instances is 0.09 seconds. Yet, the result of the SLS was optimum in 56.73% of all instances that the B&B solved to optimality within the time limit. The mean absolute percentage deviation (MAPD) to the optimum value is 0.20% (see Table 3). The runtime of the VLS is higher. With the increased runtime, more instances were solved to optimum (59.91%), with a similar MAPD value (0.17%). The runtime of the RLS is even higher. Also, it found the optimum in more cases (64.42%). However, at 0.50%, its MAPD is much higher than that of the SLS and the VLS. A cause of this effect is the randomness involved in the last algorithm. In the initial schedule, the jobs are completely arbitrarily sorted. This is useful, as more randomness in the initial schedule hits the optimum more often than with a biased sorting. But for solutions where the jobs are mostly sorted in order of their midtime, the deviation from the optimum is much higher. There, a less random initial schedule, as with the VLS, is more useful. This rationale is underlined by the fact that the MAPD of the RLS increases for the largest values of n.

The TrB&B heuristic shows low runtimes in the case of $\sigma = 6$. The number of optimally solved instances is much higher than that of the previous heuristics, as it is visible in Table 1. Still, its MAPD at 0.06% is much less. Moreover, even its runtime for $n \leq 48$ is comparable to the runtime of the VLS and the RLS. For $n > 48$, it remains within a factor of 3 of the RLS. But here, the number of solved instances remains way above about 70%, while the RLS decreases to about 40%. The TrB&B heuristic is very scalable. Depending on the parameterization, it is able to solve instances either quickly, or with a high quality. For the smaller $\sigma = 3$, its runtime is roughly half that of the B&B. The quality then is a bit higher than with $\sigma = 6$: the number of solved instances grows from 82.22% to 85.57%. In all heuristics, the number of optimally solved instances decreases for larger n. One may note that for the TrB&B, this value is increasing with $n = 60$. The source of this increase likely is caused by the correlation between the TrB&B and the B&B performance for the same instances, as well as the decrease of quality in the initial upper bound.

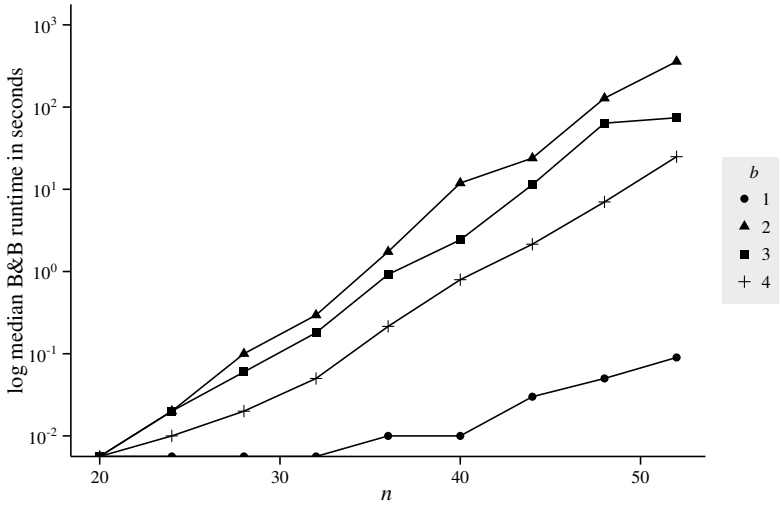

Fig. 13 Logarithmic median B&B runtime in seconds for number of jobs n and base length case b.

a	0.05	0.1	0.2	0.4	0.6
solved by B&B	82.05%	81.29%	87.05%	92.73%	99.09%

Table 4 The ratio of instances which the B&B solved to optimality within the time limit for each processing time growth factor a.

The test set allows to analyze the behavior of our algorithms in different scenarios of base lengths. The case $b = 1$ with jobs of equal base length is easiest to solve: For $n = 60$, B&B found optimum solutions for all of those instances, in a median runtime of 0.01 seconds. In this base length case, the result of the SLS always equals to that of the VLS because their initial solutions are alike. But these two achieve the optimum only in 58.79% of the instances. This result shows that solving these instances, other than one may intuitively expect, indeed involves more than simply sorting the jobs by their ideal midtime. The RLS performed better here (67.33%). For the TrB&B, this case means no difficulty (98.30% for $\sigma = 6$, and 100.00% for $\sigma = 3$). A comparison to the other base length cases is shown in Figure 13. Case $b = 2$ took the most time to solve, the base length variance is highest here. The B&B found the optimum in 78.67%. Case $b = 3$ is less hard (B&B found the optimum in 84.06%). A reason might be the following: As the lengths are not forced to be distinct, jobs more often share the same base length. In case $b = 4$, more small base lengths are drawn, hence, this case is even easier (91.03%).

The factor a specifies the growth of the processing time. The test set contains instances for several values of a. The results in Table 4 show that small values of a are more difficult to solve. With less processing time growth, the differences between different schedules are smaller. As the bounds in the B&B mostly measure processing time differences, they become hard to apply for small a. We assume that this is the reason that small values of a imply a higher difficulty to the B&B.

Concluding, we observe that the MIP is not a viable option for solving instances even with only $n = 20$ jobs. The B&B is able to solve instances of large numbers of jobs. Even for $n = 60$, most instances were solved exactly within the time limit of 480 seconds. If

heuristic solutions are acceptable, the quickest algorithm is the SLS. Its runtimes are small, and still, it solves most of all instances to optimality and, if not, with a small deviation. Both randomized variants of the SLS, the VLS and the RLS, take more time but yield better results. Although the deviation from the optimum is in general slightly higher, the RLS hits the optimum solution more often than the VLS. The TrB&B achieves the best heuristic solutions, within a low runtime.

Note that we also implemented other metaheuristics. However, a tabu search procedure and a genetic algorithm delivered results that were not as promising.

9 Conclusion

To our knowledge, the introduced DTMP is the first time dependent single machine problem which uses a nonmonotonic piecewise linear processing time function.

The analysis of several properties of the DTMP enables us to reduce the NP hard Even Odd Partition Problem (Garey et al., 1988) to the DTMP, the cDTMP, and the fDTMP. For the cfDTMP, we introduce a polynomial time algorithm. For the DTMP and the fDTMP, we present a positional assignment MIP formulation. To find exact solutions for larger instances of the DTMP, we develop several heuristics and two exact algorithms: a DP, and a B&B. The latter is modified as a sophisticated heuristic, the TrB&B. Simple heuristics, based on greedily initialized schedules, improved by a steepest ascent neighborhood search, provide an initial upper bound. A lower bound is constructed by exploiting several presented DTMP properties. Furthermore, the $C_j(t_j)$ function is monotonically increasing, it thus allows for two dominance rules that compare partial solutions in order to select the dominant one.

Future research could also consider extending the DTMP model. In this work, we consider the variant of a flexible start time to minimize the makespan, and the subproblems where all jobs have a common ideal midtime M. In a further study, one might consider to change growth factor a depending on the sign of $m_j - M_j$. This may model a worker who is walking beside the conveyor, not atop. Moreover, one could provide an individual a_j for each job j, which represents, e.g., variable walk speeds caused by materials of different size and mass. Instead of using the discrepancy of a job's midtime to the ideal midime, one could also measure the discrepancy of a job's start or completion time to a reference time. Even more general is the discrepancy of some arbitrary point in between. Each models a different point in time the worker walks. Especially the use of a job's start time as reference would be more in line with existing time dependent scheduling research. Accordingly, it is interesting to obtain knowledge about relations between these problem variants, e.g., in terms of complexity and solution methods. By this, we would gain further theoretical understanding of scheduling with convex piecewise linear processing times, and a variety of its practical applications.

Relocating the shelves at assembly lines is another way of minimizing worker paths. This can be modeled by ideal midtime assignment. The effectiveness of worker scheduling to minimize walk times is limited by the shelf positions. By allowing to change shelf positions, thus controlling ideal midtimes, we could minimize worker paths even more.

References

Ahn BH, Shin JY (1991) Vehicle-routeing with time windows and time-varying congestion. The Journal of the Operational Research Society 42(5):393–400

Alidaee B, Womer NK (1999) Scheduling with time dependent processing times: Review and extensions. The Journal of the Operational Research Society 50(7):711–720

Andrés C, Miralles C, Pastor R (2008) Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. European Journal of Operational Research 187(3):1212–1223

Bautista J, Pereira J (2007) Ant algorithms for a time and space constrained assembly line balancing problem. European Journal of Operational Research 177(3):2016–2032

Biskup D (2008) A state-of-the-art review on scheduling with learning effects. European Journal of Operational Research 188(2):315–329

Boysen N, Fliedner M, Scholl A (2007) A classification of assembly line balancing problems. European Journal of Operational Research 183(2):674–693

Boysen N, Emde S, Hoeck M, Kauderer M (2015) Part logistics in the automotive industry: Decision problems, literature review and research agenda. European Journal of Operational Research 242(1):107–120

Browne S, Yechiali U (1990) Scheduling deteriorating jobs on a single processor. Operations Research 38(3):495–498

Bukchin Y, Meller RD (2005) A space allocation algorithm for assembly line components. IIE Transactions 37(1):51–61

Bülbül K, Kaminsky P, Yano C (2007) Preemption in single machine earliness/tardiness scheduling. Journal of Scheduling 10(4-5):271–292

Cai JY, Cai P, Zhu Y (1998) On a scheduling problem of time deteriorating jobs. Journal of Complexity 14(2):190–209

Cheng TCE, Ding Q, Kovalyov MY, Bachman A, Janiak A (2003) Scheduling jobs with piecewise linear decreasing processing times. Naval Research Logistics 50(6):531–554

Cheng TCE, Ding Q, Lin BMT (2004) A concise survey of scheduling with time-dependent processing times. European Journal of Operational Research 152(1):1–13

Chica M, Cordón Ó, Damas S, Bautista J (2012) Multiobjective memetic algorithms for time and space assembly line balancing. Engineering Applications of Artificial Intelligence 25(2):254–273

Du J, Leung JYT (1990) Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 15(3):483–495

Garey MR, Tarjan RE, Wilfong GT (1988) One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Research 13(2):330–348

Gawiejnowicz S (2008) Time-dependent scheduling. Monographs in Theoretical Computer Science, Springer, Berlin, Heidelberg

Hall NG, Kubiak W, Sethi SP (1991) Earliness–tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. Operations Research 39(5):847–856

Held M, Karp RM (1962) A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics 10(1):196–210

Janiak A, Krysiak T, Trela R (2011) Scheduling problems with learning and ageing effects: A survey. Decision Making in Manufacturing and Services 5(1-2):19–36

Ji M, Cheng TCE (2007) An FPTAS for scheduling jobs with piecewise linear decreasing processing times to minimize makespan. Information Processing Letters 102(2-3):41–47

Kacem I (2010) Fully polynomial time approximation scheme for the total weighted tardiness minimization with a common due date. Discrete Applied Mathematics 158(9):1035–1040

Kanet JJ (1981) Minimizing the average deviation of job completion times about a common due date. Naval Research Logistics Quarterly 28(4):643–651

Keha AB, Khowala K, Fowler JW (2009) Mixed integer programming formulations for single machine scheduling problems. Computers & Industrial Engineering 56(1):357–367

Kellerer H, Strusevich VA (2006) A fully polynomial approximation scheme for the single machine weighted total tardiness problem with a common due date. Theoretical Computer Science 369(1-3):230–238

Klampfl E, Gusikhin O, Rossi G (2006) Optimization of workcell layouts in a mixed-model assembly line environment. International Journal of Flexible Manufacturing Systems 17(4):277–299

Kononov AV (1997) On schedules of a single machine jobs with processing times nonlinear in time. In: Korshunov AD (ed) Operations Research and Discrete Analysis, vol 391, Springer Netherlands, Dordrecht, pp 109–122

Kovalyov MY, Kubiak W (1998) A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs. Journal of Heuristics 3(4):287–297

Kubiak W, van de Velde SL (1998) Scheduling deteriorating jobs to minimize makespan. Naval Research Logistics 45(5):511–523

Kunnathur AS, Gupta SK (1990) Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. European Journal of Operational Research 47(1):56–64

Lawler EL (1977) A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1:331–342

Lawler EL, Moore JM (1969) A functional equation and its application to resource allocation and sequencing problems. Management Science 16(1):77–84

Lee CY, Vairaktarakis GL (1993) Complexity of single machine hierarchical scheduling: A survey. In: Pardalos PM (ed) Complexity in numerical optimization, World Scientific, Singapore and River Edge, NJ, pp 269–298

Malandraki C, Daskin MS (1992) Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. Transportation Science 26(3):185–200

Martino L, Pastor R (2010) Heuristic procedures for solving the general assembly line balancing problem with setups. International Journal of Production Research 48(6):1787–1804

Picard JC, Queyranne M (1978) The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. Operations Research 26(1):86–110

Scholl A, Boysen N, Fliedner M (2013) The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. OR Spectrum 35(1):291–320

Sourd F (2009) New exact algorithms for one-machine earliness-tardiness scheduling. INFORMS Journal on Computing 21(1):167–175

Sourd F, Kedad-Sidhoum S (2008) A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. Journal of Scheduling 11(1):49–58

Tanaka S, Fujikuma S, Araki M (2009) An exact algorithm for single-machine scheduling without machine idle time. Journal of Scheduling 12(6):575–593

Wan L, Yuan J (2013) Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard. Operations Research Letters 41(4):363–365

Wu CC, Shiau YR, Lee LH, Lee WC (2009) Scheduling deteriorating jobs to minimize the makespan on a single machine. The International Journal of Advanced Manufacturing Technology 44(11-12):1230–1236